FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Miika Moilanen

Topic Distiller:
Distilling Semantic Topics From Documents

Master's Thesis
Degree Programme in Computer Science and Engineering
May 2019

Moilanen M. (2019) Topic Distiller: Distilling Semantic Topics From Documents. University of Oulu, Degree Programme in Computer Science and Engineering. Master's thesis, 72 p.

ABSTRACT

This thesis details the design and implementation of a system that can find relevant and latent semantic topics from textual documents. The design of this system, named Topic Distiller, is inspired by research conducted on automatic keyphrase extraction and automatic topic labeling, and it employs entity linking and knowledge bases to reduce text documents to their semantic topics.

The Topic Distiller is evaluated using methods and datasets used in information retrieval and automatic keyphrase extraction. On top of the common datasets used in the literature three additional datasets are created to evaluate the system.

The evaluation reveals that the Topic Distiller is able to find relevant and latent topics from textual documents, beating the state-of-the-art automatic keyphrase methods in performance when used on news articles and social media posts.

Keywords: natural language processing, entity linking, automatic keyphrase extraction, automatic topic labeling, graph theory, ontology, knowledge base, information retrieval

Moilanen M. (2019) Topic Distiller: Semanttisten aiheiden suodattaminen dokumenteista. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Diplomityö, 72 s.

TIIVISTELMÄ

Tässä diplomityössä tarkastellaan järjestelmää, joka pystyy löytämään tekstistä relevantteja ja piileviä semanttisia aihealueita, sekä kyseisen järjestelmän suunnittelua ja implementaatiota. Tämän Topic Distiller -järjestelmän suunnittelu ammentaa inspiraatiota automaattisen termintunnistamisen ja automaattisen aiheiden nimeämisen tutkimuksesta sekä hyödyntää automaattista semanttista annotointia ja tietämyskantoja tekstin aihealueiden löytämisessä.

Topic Distiller -järjestelmän suorituskykyä mitataan hyödyntämällä kirjallisuudessa paljon käytettyjä automaattisen termintunnistamisen evaluontimenetelmiä ja aineistoja. Näiden yleisten aineistojen lisäksi esittelemme kolme uutta aineistoa, jotka on luotu Topic Distiller -järjestelmän arviointia varten.

Evaluointi tuo ilmi, että Topic Distiller kykenee löytämään relevantteja ja piileviä aiheita tekstistä. Se päihittää kirjallisuuden viimeisimmät automaattisen termintunnistamisen menetelmät suorituskyvyssä, kun sitä käytetään uutisartikkelien sekä sosiaalisen median julkaisujen analysointiin.

Avainsanat: luonnollisen kielen käsittely, semanttinen annotointi, automaattinen termintunnistaminen, automaattinen aiheiden nimeäminen, verkkoteoria, ontologia, tietämyskanta, tiedonhaku

TABLE OF CONTENTS

FOREWORD

This thesis was a project commissioned by Liana Technologies. I am grateful for the people at Liana Technologies for the opportunity to work on such an interesting project.

I would like to thank Ulrico Celentano for supervising this thesis. His hints and suggestions helped a tremendous amount to refine all the aspects of the final work.

Special thanks goes to Juha-Mikko Ahonen and Jarkko Haapalainen who were my supervisors at Liana Technologies during the writing of this thesis. I am also immensely grateful for my dear girlfriend Eveliina, for her unlimited support and motivation during this lengthy project. The same goes for my friends and family.

Oulu, Finland May 7, 2019

Miika Moilanen

ABBREVIATIONS AND SYMBOLS

AKE	automatic keyphrase extraction
API	application programming interface
Bi-LSTM	bidirectional long short-term memory
BoW	bag-of-words
DBPS	DBpedia Spotlight
EL	entity linking
IR	information retrieval
JSON	javascript object notation
KEA	keyphrase extraction algorithm
KPD	keyphrases per document
LDA	latent Dirichlet allocation
MR	Multipartite Rank
NER	named entity recognition
NLP	natural language processing
PR	PositionRank
RDF	resource description framework
RNN	recurrent neural network
SPARQL	SPARQL protocol and RDF query language
TD	Topic Distiller
tfidf	term frequency inversed document frequency
TPR	Topical PageRank
TR	TopicRank
XML	extensible markup language

p	precision
r	recall
f	F-score
bpref	binary preference measure
MRR	mean reciprocal rank
$\mathrm{p_R}$	R-precision

1. INTRODUCTION

Since the invention of the world wide web, the amount of freely accessible data has exploded. News articles are published at an increasing rate along with blog posts, not to mention the gargantuan amount of data produced by social media. To gain knowledge of this vast quantity of data manually is beyond the capabilities of single humans or even organizations, rendering the effort futile. To tackle the problem of making sense of this vast amount of information, automation must be exploited.

Most of the data are text-based and unstructured rendering it difficult to process them automatically. Computers are very good at processing well structured data but are lacking the ability to process natural language. Natural language processing is a discipline, closely entwined to artificial intelligence, that undertakes in the effort of augmenting computers with the ability to process natural human language. Such techniques as automatic topic modeling, summarization, and automatic keyphrase extraction are ways of distilling information from large quantities of data represented in natural human language. These techniques work as tools for humans to enhance their ability to make sense of this fast-paced world with its never-ending torrent of information.

A way to tackle the problem of making sense of the vast quantities of digital information is to bring structure to the unstructured data. This can be achieved by extracting semantic labels from the data and using these labels to index said data. Indexing the data makes it more useful since it can then be more efficiently queried. Three sub-disciplines of natural language processing have emerged to do just this: automatic keyphrase extraction (AKE), automatic topic labeling (ATL), and entity linking (EL). AKE focuses on extracting descriptive labels from single documents and ATL for collections of documents. Both AKE and ATL have researched a multitude of ways to extract essential information from textual data but the state-of-the-art suggests that employing graph theory is the way to go. This means representing text as network graphs and investigating the relationships of the nodes in those graphs. Representing text as graphs has another advantage: graphs can be connected to other graphs, such as knowledge bases. EL aims to do just this; connect documents to knowledge bases.

Semantic web [1] is a project that aims to bring structure to the world wide web. This project has given birth to large structure knowledge bases such as DBpedia [2] and Yago [3]. These knowledge bases, also known as ontologies, are reservoirs of large quantities of connected data. Taking advantages of these ontologies has proven to be highly beneficial [4], [5], [6].

This thesis details the design and implementation of a system for distilling information from textual documents: the *Topic Distiller*. Namely, the Topic Distiller is able to determine the topics discussed in a document and link those topics to DBpedia knowledge base. The topics might be explicitly mentioned in the document or they might be latent but *implied*. For example, consider the following piece of text:

Elon Musk is the CEO of Tesla.

The explicit topics in the text are *Elon Musk*, *CEO*, and *Tesla*. The latent topics would be *electric cars*, *organizations*, and *car manufacturing*.

To find the latent topics, the Topic Distiller combines EL with graph-based methods inspired by AKE and ATL. First, the EL process annotates the entities in a given text

document. Second, more general topics are found by leveraging DBpedia. Third, all the topics are combined into a network graph and graph-based measures are used to determine which are the most relevant topics.

An application programming interface (API) to the Topic Distiller is provided so that it can be deployed as a web service and used as a sub-component for larger systems.

2. RELATED WORK

This chapter describes the technologies used in the design and implementation of Topic Distiller. It also includes a literature review on the state-of-the-art of ATL and AKE to rationalize the design choices detailed in Chapter 3.

2.1. Ontologies

In philosophy, ontology refers to the theory of the nature of existence; what exists and how it exists. Artificial intelligence and web researchers have claimed the term using it to refer to a document or a file that formally defines relations between terms [1]. These ontologies serve as machine-readable sources of information, making them highly usable for automatic information processing.

Ontologies for information processing originally arose from the growing need of data integration in disciplines such as genomics, proteomics, and epidemology [7] and they were domain specific. The vision of Berners-Lee et al. [1] is to combine these ontologies into a one all-encompassing semantic web. There are several attempts at creating general, domain independent ontologies which include DBpedia [2], Freebase [8], OpenCyg [9], and Yago [3]. These general ontologies are convenient for applications such as knowledge discovery and document annotation.

Section 2.1.1 will describe in detail the DBpedia ontology, which will be used in this thesis.

2.1.1. DBpedia

DBpedia is a freely available database of structured data extracted from Wikipedia. The creators of DBpedia, namely Auer et al. [2], define it as follows: “DBpedia is a community effort to extract structured information from Wikipedia and to make this information available on the Web”.

Structure

DBpedia takes data from Wikipedia and converts it to *Resource Description Framework* (RDF) [10]. In RDF data are represented as *triples* that consist of *subject*, *predicate*, and *object*, or *resource*, *property type*, and *property value*, as depicted in parts a and b of Figure 1. The subject is always a resource, e.g. DBpedia entity. The object can be a resource or just an atomic value like a birth date, as in part c of Figure 1. The predicate or property type is the link between the predicate and object. For example, *Jay Z* is the *spouse* of *Beyoncé*, as in part d of Figure 1.

A database, such as DBpedia, that is structured in RDF format can be thought as a large network graph where all resources have properties and are connected to other resources by these properties. This allows for sophisticated analyses of relations between resources using tools such as graph theory, and social network theory.

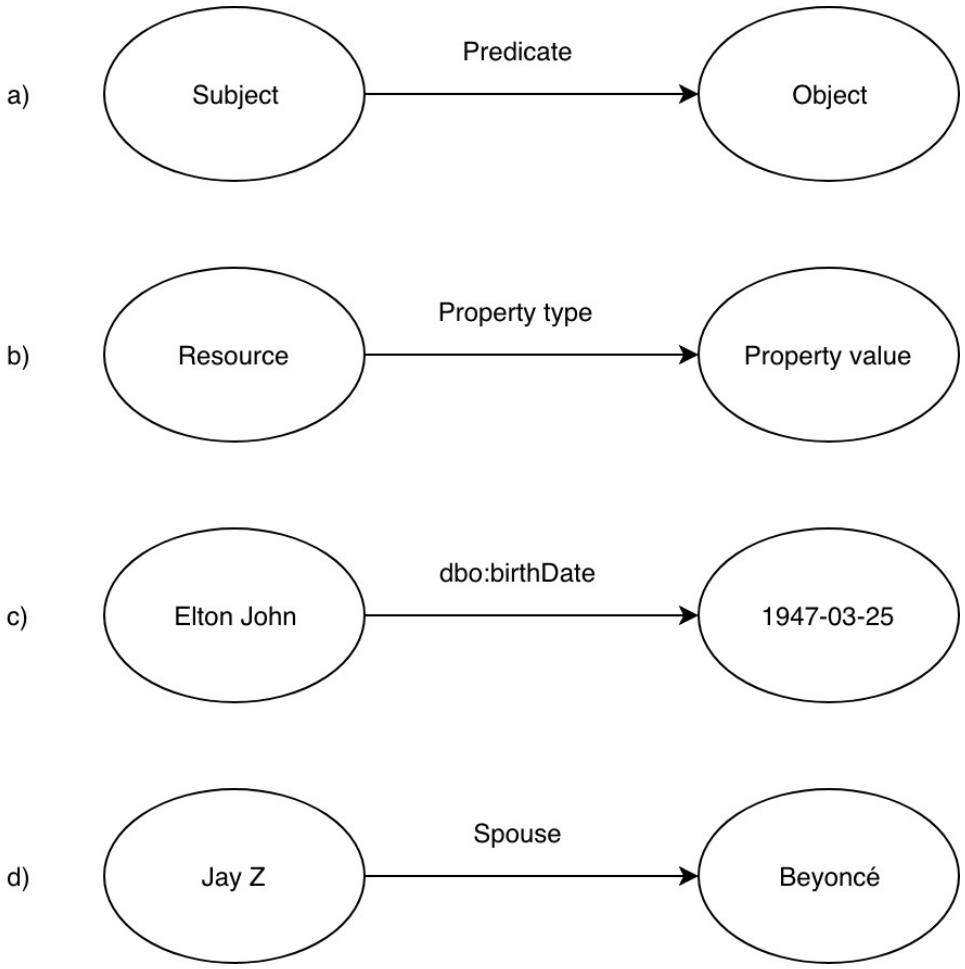


Figure 1: Example RDF triples.

Querying

To query DBpedia, a querying language is needed. SPARQL, or *SPARQL Protocol and RDF Query Language*, is recommended by the World Wide Web Consortium as a candidate query language for RDF. In its essence, SPARQL is a graph-matching query language that is used to define a pattern which is then matched against a data source [11].

The following section describes a small subset of SPARQL syntax needed to understand this thesis. A more exhaustive description of SPARQL can be read from the work of Pérez et al. in [11].

Syntax

The syntax of SPARQL is very similar to Structured Query Language (SQL). The structure of a SPARQL query is seen in Figure 2 and it consists of the following components: *prefix declarations*, *dataset definition*, *result clause*, *query pattern*, and *query modifiers* [12].

The prefix declarations part enables the usage of prefixes in the result clause making it more readable, the dataset definition defines the RDF graph that is going to be queried against (if left out, a default graph is used). The result clause specifies what

```
1  # Prefix declarations
2  PREFIX foo: <http://example.com/resources/>
3  ...
4  # Dataset definition
5  FROM ...
6  # Result clause
7  SELECT ...
8  # Query pattern
9  WHERE {
10     ...
11 }
12 # Query modifiers
13 ORDER BY ...
```

Figure 2: Structure of a SPARQL query.

kind of data is to be returned, the query pattern is the pattern that the query compares against the resources returning the ones that match, and query modifiers are used to modify the query, for example changing the ordering of the returned resources. Comments in SPARQL start with the symbol # and end at the end of the line.

An example query for the names of all the associated musical artists of Frank Zappa can be seen in Figure 3. Note that lines 5 and 6 are actually one line but they are split into two here with the separator \\ that is not part of SPARQL syntax. This is only for presentation purposes since otherwise the line would be too long.

The example query in Figure 3 starts with prefix declarations (lines 1 and 2) followed by the result clause (line 3) which starts with the command `SELECT ?name` which declares what variables from the query are to be returned. Variables are indicated by the question mark (`?`) at the beginning of their names. If one wants to return all the variables a wildcard symbol (`*`) can be used instead of the variable names.

After the `SELECT ?name` command lies the `WHERE` block (lines 4-8), both of them comprising the result clause. The `WHERE` block contains, arguably, the most important information about the query: the *triple patterns*. These patterns are used to match against the DBpedia query and all matches are returned. The first triple

```
1  PREFIX dbo:  <http://dbpedia.org/ontology/>
2  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3  SELECT ?name
4  WHERE {
5      ?artist dbo:associatedMusicalArtist \\
6        <http://dbpedia.org/resource/Frank_Zappa> .
7      ?artist foaf:name ?name .
8  }
9  ORDER BY ASC(?name)
```

Figure 3: An example SPARQL query.

is used to find all the DBpedia entities connected to the Frank Zappa entity via the `dbo:associatedMusicalArtist` property type. The first triple pattern can be translated to natural language by following the *subject* → *predicate* → *object* rule from Section 2.1.1: "Find all the *artists* that are *associated musical artists* of Frank Zappa". The second triple pattern is for attaining the names of those artists found with the first pattern. The `?artist` variable in the query is only used to realize the second triple pattern needed to get the names of the artists in question. Finally, triple patterns are terminated with periods (`.`).

The query modifier `ORDER BY ASC(?name)` (line 9) finishes the query by rearranging the results in ascending alphabetical order by the `?name` variable.

Worth mentioning is the fact that the query is missing a dataset definition. When dataset definition is omitted, the query is run against the default graph, in this case, the whole DBpedia graph itself.

The result for query in Figure 3 returned from DBpedia, in XML format [13], can be viewed in Figure 4. Note that the list is abbreviated. The resulting XML `<sparql>` element contains three attributes that describe the schema of the results. It also has two child elements, `head`, and `results`, in it. The `head` object, or the header, contains a list of the query variables. In the SPARQL query of Figure 3 only one variable *name* is specified and hence that is the sole object of the header.

The `results` object has two keys `distinct` and `ordered`. The former specifies if there are duplicates in the results and the latter indicates if the list of resources is ordered. The children of the `results` objects consist of `result` objects that are the results of the query and these in turn have `binding` objects as their children. The bindings indicate the attributes of the resources that are the variables specified in the query.

```
<sparql
  xmlns="http://www.w3.org/2005/sparql-results#"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.w3.org/2001/sw/DataAccess/rf1/result2.xsd">
 <head>
  <variable name="name"/>
 </head>
 <results distinct="false" ordered="true">
 <result>
   <binding name="name">
     <literal xml:lang="en">Adrian Belew</literal>
   </binding>
  </result>
  <result>
    <binding name="name">
      <literal xml:lang="en">Ant-Bee</literal>
    </binding>
  </result>
  <result>
    <binding name="name">
      <literal xml:lang="en">Artis the Spoonman</literal>
    </binding>
  </result>
...
  <result>
    <binding name="name">
      <literal xml:lang="en">Vinnie Colaiuta</literal>
    </binding>
  </result>
  <result>
    <binding name="name">
      <literal xml:lang="en">Warren Cuccurullo</literal>
    </binding>
  </result>
  <result>
    <binding name="name">
      <literal xml:lang="en">Wild Man Fischer</literal>
    </binding>
  </result>
 </results>
</sparql>
```

Figure 4: An example result XML from DBpedia.

2.2. Entity Linking

Having large, queryable, knowledge bases is a good step forward in bringing structure to the data residing within the world wide web. However, to use said knowledge to make sense of the unstructured textual data a method of connecting the structured with the unstructured is needed. This is where *entity linking* (EL), also known as named entity linking, entity resolution, and record linkage, comes in.

EL is defined by [14] as follows: "[EL] is the task to link entity mentions in text with their corresponding entities in a knowledge base". Entity linking is closely intertwined with *named entity recognition* (NER), which is the method of identifying *named entities* or the names of people, organizations, and geographic locations from text [15]. These named entities can, for example, be entities in a knowledge base, such as DBpedia. Here, we are going to look at the EL problem in the context of linking named entities in a document to their corresponding DBpedia entities.

NER is the first of two steps in EL, the second being *disambiguation*, which is the task of selecting the correct candidate of multiple entities sharing the same name. Disambiguation has proven to be a formidable problem in NER. For example, consider the following text:

After finishing her meal, she decided to visit Washington.

Even for a human reader, with general education, it is not self evident if the *Washington* in the text refers to the capital city of the United States, the state in the United States, or the former president of the United States. The Wikipedia disambiguation page for Washington lists over 30 cities with that name, not to mention all the persons with the same name.

The disambiguation process can be thought as a ranking problem: rank the candidates and select the topic ranking one as the linked entity. One simple approach is to rank the candidates by their popularity. For example, in case of linking entities to DBpedia the Wikipedia article view counts corresponding to the DBpedia entity could be used to rank the candidates. This method alone is able to reach 71% accuracy [16].

More sophisticated methods include context of the candidate entity into the ranking process [17]. Context of a named entity is the document it appears in and context of a DBpedia entity is its corresponding wikipedia article. Similarity between the contexts can then be used rank the entities.

2.2.1. DBpedia Spotlight

DBpedia spotlight (DBPS) is a system that links DBpedia entities to documents [18]. Originally DBPS was only capable of annotating English documents but the new, improved version [19] works with other languages, such as German, Dutch, French, Russian, and Turkish, among others. DBPS divides the task of entity-linking into three stages. The first is the *spottig stage* that recognizes phrases that may include a mention to a DBpedia entity. *Candidate selection* stage links those mentions to the DBpedia and *disambiguation* phase selects the most suitable entity among the candidates using the context around the spotted phrases.

A demo with a graphical user interface[1] is provided to test the service along with a public API[2] that can be used to programmatically annotate textual documents. The parameters for the public API for annotation can be seen in Table 1.

Table 1: DBpedia Spotlight web service API parameters for annotation

Parameter	Description
text	The text to be annotated
url	URL to be annotated
confidence	Confidence score for disambiguation
support	Number of inlinks in Wikipedia
types	Select which DBpedia types are included in the annotation
sparql	SPARQL filtering
policy	Whitelist or blacklist by type

The DBPS API comes with endpoints for spotting and candidate selection but they are outside the scope of this thesis.

2.3. Automatic Keyphrase Extraction

Automatic keyphrase extraction (AKE) is defined by [20] "as the automatic selection of important, topical phrases from within the body of a document". In other words, the purpose of automatic keyphrase extraction is to find a list of phrases that capture the topics that are related to the document in question. The terms keyphrase and keyword are sometimes used interchangeably but in this thesis the term keyphrase is preferred because the phrases usually contain multiple words instead of just one since many real world entities are named using multiple words, e.g. *artificial intelligence* or *Kim Kardashian*. The extracted keyphrases are used further to improve other natural language processing (NLP) and information retrieval (IR) tasks such as text summarization [21], text categorization [22], opinion mining [23], and document indexing [24].

Automatic keyphrase extraction usually contains two phases: *keyphrase candidate extraction* and *candidate ranking*. Keyphrase candidates are usually extracted using heuristic rules [25]. The rules are designed so that they extract only plausible keyphrases. Among the most used heuristics are stop word removal – e.g. removing all the most common words that have little semantic meaning – [26] and allowing only certain parts of speech to be used in the keyphrases, such as nouns, verbs and adjectives [27]. Other approaches include selecting keyphrases that have n-grams that appear in Wikipedia article titles and extracting n-grams and noun phrases that follow some lexico-syntactic pattern [4].

These heuristics usually work well for medium length documents, such as news articles, but for longer documents, such as books chapters, they tend to produce long candidate lists. This can be alleviated by pruning methods [28], [29], [30], [31]. For

[1] https://www.dbpedia-spotlight.org/demo/

[2] https://www.dbpedia-spotlight.org/api

shorter texts, such as social media postings, keyphrase candidate list generation might not be feasible at all since the texts themselves can be as short as a single keyphrase.

Social media postings like Facebook updates or tweets are also problematic since the language they use is more relaxed and does not always follow syntactic rules and correct spelling. Many methods like Part-of-Speech (POS) tagging and machine translation suffer from such conditions since they require context around the words for disambiguation. Automatic keyphrase extraction is no different in this regard. Many methods used to extract keyphrases utilize POS tagging and methods similar to machine translation. To overcome these difficulties methods such as topical keyphrase extraction [32], where keyphrases are extracted from topics learned from a collection of tweets, and clustering with supervised learning algorithm [33], have been proposed.

Different approaches to accomplish these tasks can be split into two main categories: unsupervised and supervised approaches. A taxonomy of AKE methods can be seen in Figure 5 and the blocks in that figure are described in Sections 2.3.1, and 2.3.2.

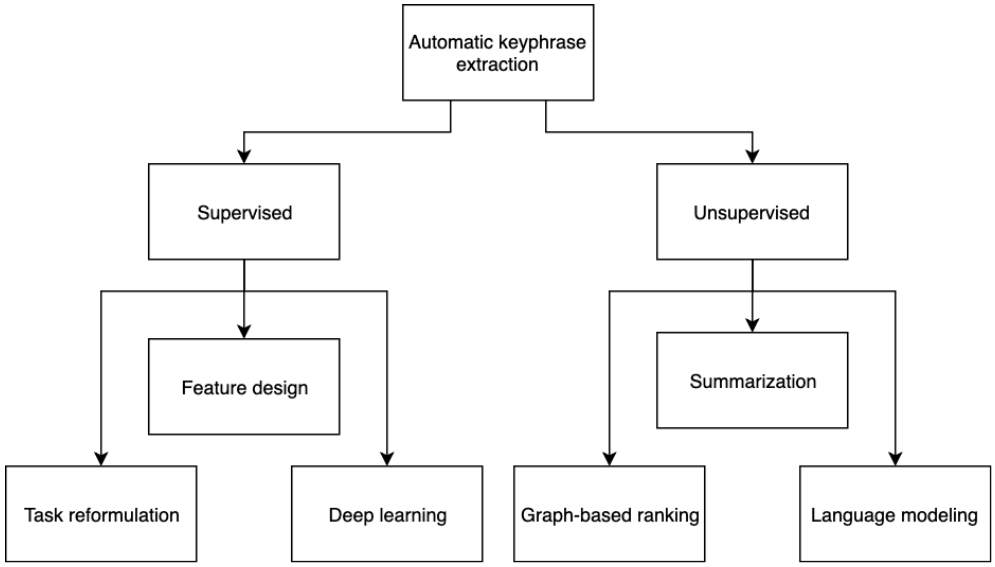


Figure 5: Tree of AKE methods.

2.3.1. Supervised Methods

Supervised methods of AKE can be divided into three categories: task reformulation, feature design, and deep learning.

Task Reformulation

Earliest attempts at supervised AKE reformulated the task as a binary classification problem. [25]. These methods include [34], and [35], where Naive Bayes learning algorithm was used, [36] where decision trees were applied, and [37] where researchers employed support vector machines. The general idea of using binary classification for AKE is to first annotate a dataset of good and bad keyphrases which can be used to train the algorithm to distinguish good keyphrases from bad.

Binary classification as an approach to AKE has one main weakness: it cannot distinguish which candidates are better than others. This makes ranking the extracted keyphrases hard. To overcome this dilemma, researchers in [37] came up with the idea of learning a ranker that can rank two candidates creating a competition between the candidates. This approach has proven to significantly outperform the popular baseline for traditional supervised AKE [35].

Feature Design

Features are designed to enhance the ability of learning algorithms to extract keywords out of documents. There are two main categories of features: *within-collection* features and *external resource-based* features [25].

Within-corpus features are embedded in the training documents themselves. They do not consult any external resources. These features include *statistical features*, *structural features*, and *syntactic features*. The most commonly used **statistical feature** is *tfidf* or *term frequency inverse document frequency* [38]. This feature captures the number of times a word appears in a document counteracted by the number of times the word appears in the corpus. The intuition behind tfidf is that words most used in a document are important words and that words that are common across the corpus are not important since documents cannot be distinguished from each other with those words. However, [39] reported no significant change in performance whether tfidf was used or not if other features were present. *Needles to say tfidf is useful only if a corpus of documents is available, which is not the case in this thesis.*

Other within-collection features include *distance* of a phrase and *supervised keyphraseness*. The former is simply the distance of the first appearance of the phrase from the start of the document [34]. The intuition behind distance is that most important keyphrases appear early in the document. Supervised keyphraseness instead is a measure of how often a keyphrase appears as a keyphrase in the training set [25]. The idea here stems from the assumption that keyphrases that are prevalent in a corpus of documents are more likely to be keyphrases in unseen documents.

All three of these statistical features can be used to form a feature set for a learning algorithm. Example of this is the keyphrase extractor algorithm KEA [35] that performs well on a variety of datasets.

Structural features use document structure to encode how suitable a phrase is as a keyphrase. For example, in scientific literature, the abstract and methods sections are assumed to include more keyphrases than other sections [40]. Another such study [41] used the title section and meta-tags of web pages to extract keyphrase candidates.

Syntactic features employ the notion that good keyphrases share a common syntactic structure. For example [41] used only noun phrases without post modification (simplex noun phrases) as keyphrase candidates. Similar methods were also utilized by [42], [43]. However, syntactic features are revealed to be unuseful if other feature types are present [41].

Outside-of-corpus features are formulated by consulting external resources, that are not included in the training corpus, such as ontologies and knowledge bases. An example of an external feature would be *Wikipedia-based keyphraseness*, which is defined by [44] to be “the likelihood of a phrase being a link in the Wikipedia corpus”. This metric is calculated by dividing the number of Wikipedia pages the phrase appears

as the anchor text of a link by the total number of Wikipedia pages containing it and multiplying the result by the number of times the phrase appears in the document.

In [41] researchers look for keyphrases in *query logs* of a search engine. The intuition behind this approach is that users use search engines to query for keyphrases they are interested in.

A different approach researched by [45] focuses on measuring how semantically related each keyphrase is instead of measuring how relevant individual keyphrases are. This leads to more coherent lists of keyphrases than approaches not employing this metric.

Deep Learning

Zhang et al. [46] compared different recurrent neural network architectures with the task of extracting keyphrases from tweets. They found out that a joint-layer recurrent neural network [47] (joint-layer RNN) performs well on this task. The joint-layer RNN has two outputs: The first one is either *true* or *false*. It predicts whether the current word is part of a keyphrase. The second output classifies the word into one of five values: *Single*, *Begin*, *Middle*, *End*, and *Not*. Single indicates that the word is a single word keyphrase. Begin, Middle, and End denote the position of the word in a keyphrase and Not indicates if the word is not a part of a keyphrase. These two outputs of the neural network are used to extract keyphrases from a document.

A similar approach is adopted by Basadella et al. [48], where instead of a joint-layer RNN they use a different kind of recurrent neural network, a bidirectional long short-term memory (herein Bi-LSTM) network. Bi-LSTM network is able to take into account the current word including its surrounding words, or put another way, its context. The Bi-LSTM Basadella et al. reported is a classifier, but unlike the joint-layer RNN of Zhang et al., the Bi-LSTM only classifies each word to one of three classes: NO_{KP}, $BEGIN_{KP}$, $INSIDE_{KP}$, which correspond to the word not belonging to a keyphrase, the word being a begin of a keyphrase or the word being inside of a keyphrase, respectively. This method surpasses a state-of-the-art method, namely TopicRank [49] (described in Section 2.3.2), in recall and F-score (see Section 2.3.3) measured against the Inspec [43] dataset but falls behind in precision.

Meng et al. [50] take a different approach to AKE. Instead of labeling each word of a document, they use an encoder-decoder RNN to generate keyphrases. Meng et al. named the method *deep keyphrase generation*. This type of an RNN was first developed by Cho et al. [51] and Sutskever et al. [52] to tackle the problem of machine translation. An interesting improvement to previous models deep keyphrase generation introduces is its ability to generate keyphrases that are not present in the document. This is because the neural network can "memorize" information of the training data giving the ability to incorporate knowledge outside of the document in question.

The problem with the deep learning techniques discussed above is that they need a large corpus of annotated training data to work well. Unfortunately, the only freely available, big datasets for AKE are based on scientific literature, making models trained on these datasets work well on scientific articles but poorly on other types of documents.

2.3.2. Unsupervised Methods

Methods used for unsupervised AKE can be split into three main categories: graph-based ranking, summarization, and language models.

Graph-based Ranking

Graph-based methods for AKE employ graph theory to rank keyphrases. The idea is to form a graph where the nodes are represented by the keyphrase candidates and edges represent the connections between those candidates. Weights of the edges encode the strength of each connection. These edges between the nodes can be thought as "votes" that rank the keyphrases by their importance. Then the top five to fifteen candidates are selected as keyphrases for the document. For example, TextRank [27] is an algorithm for AKE derived from Google's PageRank [53] algorithm that is used to rank web pages instead of keyphrases. Essentially TextRank employs the notion of PageRank that an important node in the graph is connected to a large number of other nodes and to nodes that are important themselves. [54] present PositionRank algorithm which augments the TextRank by using a version of PageRank that takes into account the position of each word in the document. Another version of TextRank, called ExpandRank [55] encodes information of neighboring documents to employ more information on the keyphrase extraction task. *This method of using neighboring documents is only viable if a corpus of related documents is available.*

Researchers have used different methods to build a graph from text, for example by connecting words that co-occur within a window of N words [27] or by connecting all candidates to each other forming a complete graph and then weighing the edges according to their semantic relatedness, like in the TopicRank method [49]. In [56], researchers use a multipartite graph instead of a conventional graph.

According to [57] graph-based models suffer from two disadvantages:

1. Graph-based models rank high keyphrases that are strongly connected. This does not ensure that they are relevant to the main topics of the document.

2. Ideally, a good set of keyphrases should contain all the main topics of the document. Graph-based methods do not guarantee this.

The weaknesses of graph-based methods can be overcome with grouping the candidate keyphrases into topics in a way that all the topics cover all and only those keyphrase candidates that are related to that topic [25]. Topic-Based Clustering aims is a way to achieve this.

KeyCluster [58] clusters semantically similar candidates using Wikipedia and co-occurence based statistics. Wikipedia is employed by computing a metric called *Wikipedia-based term relatedness*, which measures the similarity of tfidf vectors of Wikipedia articles against the tfidf vectors of keyphrase candidates. The idea is that each of these clusters represents all the topics present in the document and thus guarantees that all topics are presented in the extracted set of keyphrases. The most central nodes of the clusters are then extracted as keyphrases representing that particular topic.

Even though KeyCluster outperforms baseline methods [43] and [27], it suffers from the problem that all topics are treated equally. Ideally, only important topics should be represented as keyphrases.

Topical PageRank (TPR) [57] is a successful attempt to alleviate the problems of KeyCluster. The approach employs topic modeling as a way to incorporate out-of-corpus knowledge to enhance keyphrase extraction. First, a word topic distribution is computed using the unsupervised learning method *latent Dirichlet allocation* (LDA), which is a "generative probabilistic model for collections of discrete data such as text corpora" [59]. The idea is to represent documents as "random mixtures of over latent topics, where each topic is characterized as by a distribution of over words". The following generative process is performed for each document w in a corpus D:

1. Choose $N \sim Poisson(\xi)$
2. Choose $\theta \sim Dir(\alpha)$
3. For each of the N words w_n:
   (a) Choose topic $z_n \sim Multinominal(\theta)$
   (b) Choose word w_n from $p(w_n|z_n, \beta)$, a multinomial probability conditioned on the topic z_n.

The symbols in the preceding process are defined as follows:

- w_n is the nth word in a sequence,
- z_n is the nth topic in a set of topics,
- $Dir(\alpha)$ is the Dirichlet distribution [60], and
- β is a $k \times V$ matrix, where k is the dimensionality of the Dirichlet distribution, and V is the number of words in the vocabulary.

After computing the LDA for TPR, a word graph is built using word co-occurences. Then, a biased version of PageRank is run for each word with respect to each topic to attain importance scores of the words. Finally, the topics of the document in question and the topic-specific rankings of the words are used to get the ultimate ranking of keyphrases. Top candidate keyphrases are then selected as the final extracted keyphrases.

TPR outperforms both tfidf and TextRank significantly but the authors in [57] did not compare it with KeyCluster.

The main disadvantage of TPR is that it is very computationally expensive. This is because the PageRank itself is a costly algorithm to run and it has to be run for each topic in the model. [57] report that best results were obtained with 500 topics, meaning that PageRank should be run 500 times per document. This may become infeasible if high throughput is required.

To mitigate the computational cost Sterckx et al. [61] propose a modification to TPR that runs PageRank only once per document. Sterckx et al. are able to reproduce

near equal performance compared to original TPR by using a single value to represent topical word importance instead of using a value per topic in the model.

CommunityCluster [62] is another clustering method that like TPR assigns more significance to important topics, but unlike TPR, extracts all keyphrses from an important topic instead of one. This is based on the assumption that candidates that receive only little focus are important to the document if they are associated with important topics. CommunityCluster outperforms TextRank and tfidf in recall while not sacrificing precision.

Summarization

Extracting keyphrases from a document is very similar to summarizing a document. Mani and Inderjeet [63] define the goal of automatic text summarization as "to take an information source, extract content from it, and present the most important content to the user in a condensed form and in a manner sensitive to the user's or application's needs". A more practical definition is given by Das et al. [21]: "Summaries may be produced from a *single document* or *multiple documents*, summaries should preserve important information, and summaries should be short."

Researchers have come up with approaches that combine AKE and automatic summarization by learning a model that achieves both simultaneously. Zha and Honguyan [64] use graph-based methods to compute saliency scores for words and sentences. The intuition behind Zha's and Honguyan's work is that important words appear in important sentences and vice versa.

Wan et al. [65] expand on Zha's and Honguyan's work by two additional assumptions: An important sentence is connected to other important sentences, and an important word is connected to other important words. Wan et al. use these assumptions to build three weighted graphs, namely sentence-to-sentence (S-S) graph, word-to-word (W-W) graph and a bipartite sentence-to-word (S-W) graph. The weights of edges in S-S graph represent the similarity between the two sentences. In W-W the weights correspond either to co-occurence of the words or to knowledge-based similarity. S-W graphs edge weights stand for the importance of the word in the sentence it appears in. In this manner, Wan et al. combine the strengths of Zha's work with the ideas of TextRank outperforming both of them [25].

Language Models

As mentioned in Section 2.3, conventional AKE approaches use a two-staged approach: keyphrase candidate selection and keyphrase candidate ranking. Tomokiyo and Hurst [66] have devised a method that combines these two stages. They achieve this by scoring keyphrases for *phraseness* and *informativeness* based on statistical language models. *Phraseness* denotes how likely a word sequence is a phrase and *informativeness* describes how well a phrase encapsulates the important ideas of a given document. These two features are combined to rank the keyphrases.

The language models used in denoting the phraseness and informativeness values are trained on a *foreground* corpus and a *background* corpus. The foreground corpus consists of the documents the keyphrases are to be extracted from, and the background

corpus of documents that are not used for keyphrase extraction (e.g. a collection of web pages) that encapsulate general information about the world.

Language models called *word embeddings* have become important workhorses in NLP applications after word2vec was introduced by Mikolov et al. [67] in 2013. Traditionally documents have been represented as bag-of-words (BoW) models. BoW encodes the words of a document as a binary or frequency based vectors [68]. The conventional BoW model represents words as one-hot-vectors wherein each word corresponds to distinct vector component [69]. Let's illustrate this with an example:

The quick brown fox jumped over the lazy dog.

Let's split that into tokens, remove duplicates, and sort alphabetically

$$C = \{brown, dog, fox, jumped, lazy, over, quick, the\} \tag{1}$$

C can be represented as a matrix of one-hot vectors like this:

$$C\text{`} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2}$$

where each row represents a word, e.g.:

$$brown = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3}$$

and

$$lazy = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

There are some problems with representing words as one-hot vectors. First, this representation creates a sparse vector space with a dimensionality that can get very big. The small vocabulary in Expression 1 consists only of eight words, but consider a larger vocabulary of 50 000 or 50 000 000 words. The dimensionalities would be 50 000 and 50 000 000 respectively. Second, representing words with one-hot vectors does not capture their semantic meaning since there is no good way to compare words with one another.

Word2vec alleviates the problems of BoW by both reducing the dimensionality and capturing the semantic meaning of the words. It uses the latent space of a neural network to encode semantic information about words in a multi-dimensional vector space. Words are represented as dense vectors. Semantically similar words appear close to each other in the vector space and the semantic difference between two words can be computed with cosine similarity [67]. Other related approaches include GloVe [70] and Fasttext [71].

A modification to word2vec by Le and Mikolov [72] encodes entire phrases or documents into dense vectors. This allows for the comparison of entire documents

with each other. A similar approach, key2vec, is adopted by Mahata et al. [73] to tackle AKE. Mahata et al trained a multi-word phrase embedding model using Fasttext to assign a *theme vector* to a document. This theme vector is constructed by first extracting a *theme excerpt* from a given document. The theme excerpt consists of the first few phrases of the document. The pre-trained Fasttext model is then used to convert this excerpt to a theme vector. Keyphrases candidates are then scored by their cosine similarity to the theme vector. key2vec surpasses Topical PageRank [57] and TopicRank [49] in performance on the SemEval2010 dataset (see Section 4.1.1).

In a similar approach, called EmbedRank, Bennani-Smires et al. [74] embed both the candidate keyphrases and the document itself into the same vector space. The candidate keyphrases are scored by their cosine distance to the document embedding. EmbedRank outperforms TextRank and TopicRank, among others, when using Inspec [43] and DUC2001[3] datasets, but loses to TopicRank when measured with NUS [40] dataset.

2.3.3. Evaluation of Automatic Keyphrase Extraction Techniques

Evaluation of AKE methods is hard. The conventional way is to use a dataset of documents with manually extracted, *gold standard*, keyphrases and compare those to automatically extracted ones. The prevalent method of comparison is using exact matches, meaning that the automatically extracted keyphrases should be exactly the same as the manually extracted ones. Usually, measurements focus on precision (p), recall (r), and F-score (f), which are defined in the equations 5, 6, and 7 respectively:

$$\mathrm{p} = \frac{c_{\mathrm{correct}}}{c_{\mathrm{extract}}}, \tag{5}$$

$$\mathrm{r} = \frac{c_{\mathrm{correct}}}{c_{\mathrm{standard}}}, \tag{6}$$

$$\mathrm{f} = \frac{2\mathrm{pr}}{\mathrm{p} + \mathrm{r}}, \tag{7}$$

where c_{correct} is the total number of correct keyphrases extracted by the system, c_{extract} is the total number of automatically extracted keyphrases, and c_{standard} is the total number of human annotated keyphrases [57].

This way of evaluating the performance of an AKE system is not ideal. Consider the following example keyphrase *Neural network* and compare it to *Artifical neural network* [25]. Both keyphrases mean the same thing but the latter is a bit more precise. Consider then that the former is a manually extracted, gold standard keyphrase and the second is automatically extracted. Using exact matches, this pair would be considered as a failure to extract a keyphrase successfully. Even though using p, r, and f is less than ideal, these metrics are used widely in the literature to compare different AKE methods. For this reason, these metrics are used in this thesis too.

Some studies have used human evaluation [74], [75], but it is time-consuming and expensive making it unsuitable for parameter tuning.

[3] https://www-nlpir.nist.gov/projects/duc/data.html

Multiple solutions to alleviate the problems of exact matching have been proposed. Liu et al. [57] used *binary preference measure* (bpref) [76] to evaluate the ranking of keyphrases. For a topic with R relevant documents where rd is a relevant document and n is a member of the first R judged nonrelevant documents as retrieved by the system [76],

$$\mathrm{bpref} = \frac{1}{R}\sum_{\mathrm{rd}} 1 - \frac{|\mathrm{n\ ranked\ higher\ than\ rd}|}{\mathrm{R}}. \tag{8}$$

Essentially, bpref measures what portion of the correctly matched keyphrases is ranked higher than the non-correct ones.

Another measure Liu et al. used was *mean reciprocal rank* (MRR) [57]. MRR measures how the first correctly extracted keyphrases by a system are ranked. For a document d, rank rank_d is denoted as the position of the of the first correct keyphrase in the list of all extracted keyphrases,

$$\mathrm{MRR} = \frac{1}{|\mathrm{D}|}\sum_{\mathrm{d}} \in \mathrm{D}\frac{1}{\mathrm{rank}_d}, \tag{9}$$

where D is the set of documents for keyphrase extraction.

Datasets usually have a variable number of keyphrases extracted from documents. This means that AKE systems that extract more, albeit correct, keyphrases that are annotated in the gold standard will have poor performance measured with exact matching. For example, if a system always extracts exactly 10 keyphrases and the gold standard always has 8 keyphrases, two of the automatically extracted keyphrases will be erroneous [77]. To tackle this problem Zesch and Gurevych [77] propose usage of information retrieval metric *R-precision* (p_R). Zesch and Gurevych define $\mathrm{p_R}$ for AKE as follows: $\mathrm{p_R}$ "is the precision when the number of retrieved keyphrase matchings equals the number of gold standard keyphrases assigned to the document". Formally:

$$\mathrm{p_R} = \frac{c_{\mathrm{correct}}}{c_{\mathrm{standard}}},\ \text{when}\ c_{\mathrm{standard}} = |\mathrm{K_{gold}}|, \tag{10}$$

where $\mathrm{K_{gold}}$ is the set of gold standard keyphrases. This ensures that the possible mismatch between number of extracted and gold standard keyphrases does not affect the measurement.

Zesch et al also introduce an *approximate matching strategy*, where morphological variants of keyphrases are accepted as matches along with keyphrases that include a gold standard keyphrase and keyphrases that are a part of a gold standard keyphrase. This approximate matching strategy is further detailed in section 4.2

2.3.4. Shortcomings of Automatic Keyphrase Extraction

Hasen et al. [25] present an error analysis on most popular AKE techniques. The problems are summarized here.

Overgeneration errors happen when a system correctly extracts a keyphrase because it contains a common word in a document but then erroneously selects other keyphrases with the same word. **Infrequency errors** happen when the system is not recognizing a keyphrase due to its infrequent appearance in the document in

question. **Redundancy errors** occur when a system predicts two or more semantically equivalent keyphrases.

Another problem with AKE is that the conventional methods are unable to extract keyphrases successfully on short texts such as tweets and other social media postings. Some methods have been suggested to tackle this problem, such as augmenting KEA [35] with Brown clustering and continuous word vectors and feature engineering combined with gradient boosting machine [78]. These methods show small improvements on baseline methods.

Lastly, none of the AKE methods discussed in Sections 2.3.1, and 2.3.2 are guaranteed to produce keyphrases that represent real world phenomena. Instead, by using lexical methods to combine words into keyhprase candidates they all can potentially generate nonsensical keyphrases. This is unacceptable for the purposes of this thesis, hence none of these methods can be used as is for building the Topic Distiller.

2.4. Automatic Topic Labeling

Topic modeling is a way to identify the subject matter of a collection of documents by determining its inherently addressed topics [6]. Simply put, a topic model is a collection of terms explicitly appearing in a set of documents and their corresponding marginal probabilities to appear in a given topic. Marginal probability is defined as the probability of a term to appear in a topic. These topic models are used for various NLP tasks, such as multi-document summarization [79], sentiment analysis [80], and word sense induction [81], among others.

Most used techniques for topic modeling include LDA [59], *latent semantic analysis* (LSA) [82], and *probabilistic latent semantic analysis* (PLSA) [83].

The problem with topic modeling is that the resulting topics are hardly interpretable by humans. One standard way of making sense of a topic is to extract the top ten terms with the highest probability. This results in term lists like the following:

stock, market, investor, fund, trading, investment, firm, exchange, companies, share.

With a little inspection, it is easy to see that the above list of terms belongs under a topic of *stock market trading* [84]. *Automatic topic labeling* (ATL) is a research area which focuses on finding terms such as *stock market trading* for a corresponding topic. The idea is to alleviate the cognitive load for humans to interpret raw topic term listings by explicitly identifying the semantics of the topic [84].

2.4.1. Non-graph-based Methods

Mei et al. [85] introduced the first method for ATL. The method involves extracting bigram collocations from the document in question and then ranking them based on Kullback-Leibler (KL) divergence. Since this method is fully extractive, the underlying assumption is that all relevant labels for a topic can be found from the document being topic modeled. Mei et al. report that their method performs better than the baseline method of selecting high probability words as topic labels.

Lau et al. [84] take a more abstractive approach by incorporating outside knowledge from English Wikipedia. First, Lau et al. use the top-10 topic terms to query Wikipedia using the native search API and Google's site-restricted search and take the top-8 article titles from each source. Next, secondary labels are produced by chunking the titles into words and generating all component n-grams while filtering out the n-grams that are not themselves titles of Wikipedia articles. Then, secondary labels that are stopwords or unigrams that are only marginally related to the document are removed using RACO (Related Article Conceptual Overlap) lexical association method [86]. The primary and secondary labels are then ranked using supervised (support vector regression) and unsupervised methods based on lexical association features. Lau et al. report that their method performs better than the one proposed by Mei et al. [85].

Bhatia et al. [87] employ neural embeddings word2vec (see Section 2.3.2) and its extension doc2vec [72], along with Wikipedia for ATL. The main idea is to map topic terms, represented as neural embeddings, to Wikipedia article titles. Then the best topic labels can be found simply using cosine similarity to compare embeddings. The neural embeddings are trained using a downloaded version of Wikipedia, but after the training is done, the system proposed by Bhatia et al. operates completely offline cutting out the need for third-party services. Bhatia et al. report that their system beats the system proposed by Lau et al. [84] both in performance and simplicity.

2.4.2. Graph-based Methods

Allahyari et al. [5] take a different approach. They, unlike Mei et al. and Lau et al., modify the underlying algorithm that produces the multinomial topics, namely LDA. The method proposed by Allahyari et al. integrates ontological concepts with topic models, by representing each topic as a multinomial distribution over *concepts* rather than simple words. These concepts are then represented by a multinomial distribution over words. Allahyari et al. call this method *OntoLDA*.

After generating the topics, with OntoLDA, a *semantic graph* is constructed. This semantic graph is built by extracting the top ranking concepts of a document, as vertices of the graph, based on their marginal probability then connecting these vertices by edges if they are related in the DBpedia ontology. The semantic graph consists of subcomponents called *thematic graphs* which are the connected components of the semantic graph. Then the thematic graph with most connections is selected as the *dominant thematic graph* and its nodes are ranked using the HITS (Hyperlink-Induced Topic Search) algorithm [88], also known as the *Hubs and Authorities* algorithm. These highest ranking nodes are the *core concepts* of the topic being labeled. Further, a *topic graph* is extracted by traversing DBpedia at most three hops away from each core concept and the union of these topic graphs is called the *topic label graph*.

The core concepts of a label are treated as the topic label candidates. Each of these candidates is scored and ranked using three metrics: *membership score*, *coverage score*, and *semantic similarity* explained in detail in [5].

The intuition behind the method proposed by Allahyari et al. is that ontology concepts and entities occurring in the document better determine the document's topics than simple words. This makes sense since ontology concepts and entities carry more

semantic meaning than simple, ambiguous words. Allahyari et al. report that their measure surpasses the method proposed by Mei et al. [85] by a large margin.

Similar, albeit simpler, to the approach developed by Allahyari et al. is a method devised by Hulpus et al. [6]. This method also incorporates the DBpedia ontology in constructing a graph which is used to label topics. Instead of proposing their own algorithm for topic extraction, Hulpus et al. use the conventional LDA to produce topic terms that are then disambiguated using a word sense disambiguation (WSD) method proposed by Hulpus et al. [89] in their previous work. Said WSD method determines a set of DBpedia entities, which Hulpus et al. call *concepts*, where each concept is a sense of one of the top terms provided by LDA. For each extracted concept a *sense graph* is constructed by traversing the DBpedia ontology. Hulpus et al. report that DBpedia entities at most two hops away from the concepts are used to build the graph since taking more hops will produce very large graphs with a lot of noise. The union of the produced sense graphs is called the *topic graph* and the concepts acquired with WSD are called the *seed concepts* of that graph. The intuition behind this approach is that concepts of a topic are related and thusly they should lie near each other in the DBpedia graph forming one connected graph.

After the topic graph is attained, suitable topic labels are extracted using a network centrality measure. The assumption is that the most important concepts in the topic graph are the central ones. Hulpus et al. compare their method implemented with two centrality measures *focused information centrality*, and *focused random walk betweennes centrality*, with the one proposed by Mei et al. The results show that Hulpus' and Greene's method performs better.

Aletras and Stevenson [90] propose yet another graph-based method for ATL. Instead of relying in an ontology, like [5] and [6], they use information obtained from web searches. The topic label candidate extraction Aletras and Stevenson use is the one proposed by Lau et al. [84]. A web search is then leveraged in identification of most salient labels by querying a search engine with the top terms produced by a topic modeling algorithm. The metadata of the search results has a list of keywords that are used to construct a graph following the TextRank method for automatic keyphrase extraction [27]. Like with TextRank, PageRank is then utilized for ranking the topic labels and top ranked ones are selected as the actual labels. Aletras and Stevenson report their proposed method surpasses the one proposed by Lau et al. [84] measured with the same metrics used by Lau et al. in their paper.

2.5. Discussion

Both AKE and ATL have similar objectives and methods to accomplish the said objectives. AKE is conventionally used to find keyphrases from a single document where topic modeling with ATL focuses on distilling information of document collections into phrases that capture their core ideas. Both of these objectives, in their core, aim to decrease the cognitive load of human beings trying to make sense of vast quantities of textual data. To tackle the difficult problems faced by both of these research fields, researchers have used methods, such as graph theory and word embeddings, among others.

The trend in both AKE and ATL is to move from simple extractive methods to more abstractive methods. Just like human annotators, methods using external knowledge are able to find the broader topics related to the documents they process.

Further strengthening the case for using knowledge bases, Hasan et al. [25] suggest that incorporating background knowledge to AKE systems might be able to alleviate overgeneration, infrequency, and redundancy errors dicussed in Section 2.3.4.

AKE methods are commonly divided into supervised and unsupervised methods. Supervised methods perform well if they are fed a large quantity of good quality data. This means that if no such data is available the results are suboptimal, making supervised methods impractical for many use cases. No redemption is gained by the fact that most freely available datasets for AKE contain only scientific articles. Unsupervised methods do not need any training data by definition but they require more human ingenuity to bring in good results. A lot of effort by researchers has been aimed to making these unsupervised methods better and all of the state-of-the-art methods, excluding methods employing deep learning, are unsupervised. Deep learning would be a valid method to consider if building a large dataset was in the scope of this thesis.

Most of the unsupervised methods for AKE in the literature are graph-based. This attribute makes them a natural match for graph-based knowledge bases, such as DBpedia. AKE methods use different lexical features to build graphs, as discussed in Section 2.3.2. One problem with this approach is that sometimes the keyphrases extracted from the document are nonsensical; they do not represent any real word phenomena.

In this thesis, it is argued that instead of using lexical features to build these graphs it is better to employ EL to find all the mentions of knowledge base entities from a text document and build the graph using the predicates connecting those entities. In other words, the knowledge base entities would be the nodes in the graphs and the predicates connecting those entities would be the edges. Building the graph in this manner is inspired by the ATL methods developed by Allahyari et al. [5] and Hulpus et al. [6] discussed in Section 2.4.2. However, Allahyari et al. and Hulpus et al. do not employ EL to get the nodes. Instead, they rely on topical analysis.

Using EL with knowledge bases ensures that all the keyphrases, or topics, found in the document would be existing, real world topics, e.g. when using DBpedia as the knowledge base the topics found would be Wikipedia article titles. EL has the disadvantage over lexical methods in that it lacks the ability to extract new or obscure keyphrases that can not be found from the knowledge base. On the other hand, connecting documents to a knowledge base using EL enables finding latent topics by traversing the entities in the knowledge base to generate a larger graph that comprises of entities not present in the document, but closely related to it.

3. DESIGN AND IMPLEMENTATION

The goal of this thesis is to alleviate the heavy load of analyzing vast quantities of textual data. To reach said goal, a system able to extract both apparent and latent semantic topics from textual documents ranging from news articles to social media postings, is designed. This system, named the Topic Distiller, will enable application developers to build applications such as text summarizers and semantic search engines that further help professionals such as researchers, journalists, and marketeers in their daily work.

The system architecture described in this section is for an application that provides a programmable interface to enable communication with other services, such as graphical user interfaces or document indexers. This chapter first identifies the requirements for the system (see Section 3.1), followed by design principles (see Section 3.2), and then moves on to describe the composition of said system (see Section 3.3).

3.1. Requirements

The aim of this thesis is to design and implement a tool that can distill a list of topics from text documents, hence the name Topic Distiller. The implemented application also has to be able to detect topics that are latent in the document so it has to have knowledge of a large database of different topics. On top of that, the Topic Distiller has to provide an API for querying all available topics. These topics could then be further used to index documents for semantic search. The indexing is outside of the scope of this thesis.

An important use case for the our system is to identify the topics trending in social media. Dealing with social media posts is hard since they are usually short and do not follow the grammar rules strictly. The frequency at which new data are produced by social media is quite high, posing another requirement for the Topic Distiller: *high throughput*. This means that the system should be fast enough to be usable in real time social media analysis.

The sets of topics discovered by the Topic Distiller must be usable as they are to minimize manual work; no human work should be required for checking that the topics are correct. This means that the topics must represent real world phenomena and can not be nonsensical.

Ideally, the final system will have as little code as possible to be maintained, because more code will result in higher maintenance overhead and error probability.

On top of all the above requirements, the system must provide an simple interface for it to be easily integrated to other systems.

The requirements are summarized in the following list:

1. The Topic Distiller must be able to identify relevant and latent topics in a textual document.

2. The Topic Distiller must be able to extract topics from short texts such as social media postings.

3. The topics discovered by the Topic Distiller must represent real world phenomena.
4. The Topic Distiller must be able to process enough documents per minute.
5. The Topic Distiller should be easy to maintain.
6. The Topic Distiller must provide an simple interface for communicating with other systems.

3.2. Design Principles

This section describes the design principles adopted to ensure that the requirement of easy maintenance (see Section 3.1) is met.

To fulfill said requirement, a modular design is adopted. Each module is responsible for doing just one job and doing it well. The modules consist of a set of functions that implement the necessary functionality of the modules. To ensure encapsulation, interfaces between these modules, and functions, should be built in such a manner that swapping the internals of one module would not require changes in other modules. This modularity will provide for easy testing and thus reduce the number of software bugs. The system will also be stateless further diminishing the complexity of the system.

Reinventing the wheel is a fool's errand; free and open-source software will be used extensively. This will reduce the amount of maintainable code in the system by a great degree.

The design principles applied in the implementation of the Topic Distiller are summarized in the following list:

1. Modularity
2. Statelessness
3. Use of open-source software

Python was chosen as the programming language since it provides good support for the design principles listed above. Python makes it easy to organize software into self-containing modules[1], Python is fully open-source, and has a large community providing quality libraries for scientific computation.

3.3. System Design

This section describes the design of the Topic Distiller, which consists of four internal modules and two external modules. The block diagram of the designed system, seen in Figure 6, contains the internal (squares inside of the dotted line) and external (squares with round corners outside of the dotted line) modules along with the communication

[1] https://docs.python.org/3/tutorial/modules.html

between said modules. Internal modules consist of software that is specially built for the Topic Distiller. External modules are third-party services that exist independently of the Topic Distiller.

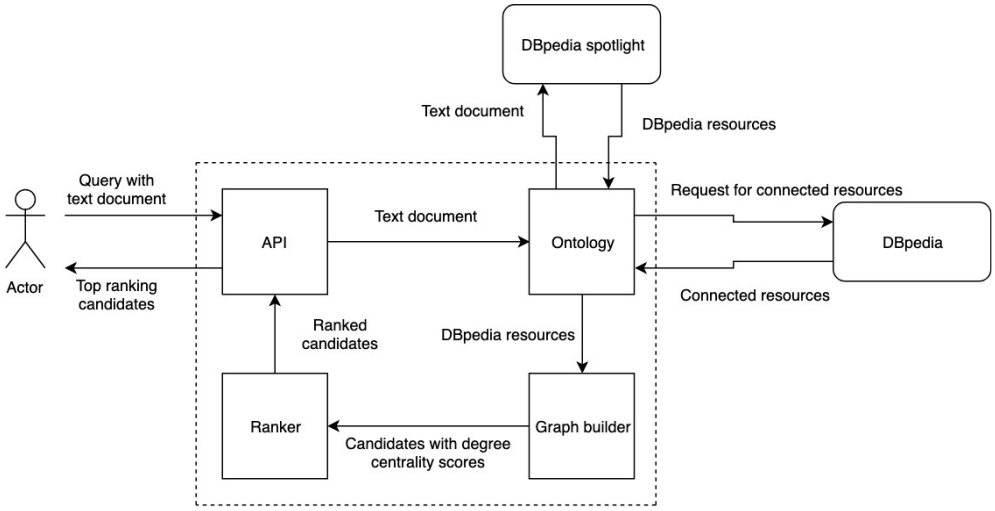


Figure 6: Block diagram of the system with its internal and external modules.

The main flow of the Topic Distiller is as follows: First, the system is provided a text document. Second, EL is used to find all the DBpedia entity mentions from said document. Third, a graph is built. Fourth, a centrality measure is used along with lexical analysis to rank the nodes, and fifth, the top ranked nodes are returned. It should be noted here that no lexical methods are used to generate the list of topic candidates, only pure EL. Lexical methods are only used to rank the already generated candidates.

Different parts of the application flow are delegated to different modules. The API module is responsible for serving the functionality of the system to the outside-world, satisfying the requirement number 6 (see Section 3.1). The ontology module employs EL to connect the document to DBpedia. Employing EL guarantees that topics represent real world phenomena, hence meeting the requirement number 3. The graph is built by the graph-module connecting DBpedia entities found in the document to more abstract DBpedia entities. This method of building the graph aims to satisfy the requirements 1 and 2.

In a famous paper, Donald E. Knuth said that “premature optimization is the root of all evil” [91]. To follow this advice no special emphasis was dedicated to designing as computationally fast system as possible. The design decision applied here is to first meet the other requirements and then evaluate if the Topic Distiller needs optimization.

The following sections 3.3.1, 3.3.2, 3.3.3, and 3.3.4 will describe the design of the modules in more detail.

3.3.1. Application Programming Interface (API)

API provides a way of communication between the system and other systems. Users of the Topic Distiller send a request to the API to extract topics with the document that the topics are to be extracted from. The document is then sent to the Ontology module, described in Section 3.3.2, for processing.

```
1 {
2     "text": "Software is a gas; \\
3              it expands to fill its container.",
4     "num_topics": 3
5 }
```

Figure 7: Example request JSON.

```
1 {
2     "topics": [
3         "Software",
4         "Computer science",
5         "Nathan Myhrvold"
6     ],
7     "success": true
8 }
```

Figure 8: Example response JSON.

The request is sent to the API using the Hyper Text Transfer Protocol's (HTTP) [92] POST method. The payload of the POST method is a JavaScript Object Notation (JSON) [93] object containing two keys: `text` and `num_topics`. The value of the former is the text, from which the topics are to be extracted, and the latter is the number of topics to be extracted. An example request JSON object is as seen in Figure 7. Note that the lines 2 and 3 of the figure are actually one line split into to using the separator `\\`. This is only for presentation purposes. Valid JSON does not permit such formatting.

The response sent from the API is also in JSON format and contains two keys, `topics`, whose value is a JSON array of text strings representing the extracted topics, and `success`, whose value is boolean `true` or `false`, indicating the success or failure of topic extraction respectively. An example response JSON can be seen in Figure 8.

If the request is malformed, or an error occurs while processing the request, the `success` key will have `false` as its value and an error message is provided along with the corresponding HTTP error code. An example JSON of failed request can be seen in Figure 9.

```
1 {
2     "success": false,
3     "error_message": "Malformed request.",
4     "error_code": 400
5 }
```

Figure 9: Example failure response JSON.

```
    {
        "text": "You're a wizard, Harry!",
        "confidence": 0.5,
        "support": 20
    }
```

Figure 10: Example of a request JSON to DBpedia Spotlight with parameters.

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dct: <http://purl.org/dc/terms/>
SELECT *
WHERE {
    OPTIONAL { <%s> skos:broader ?broader } .
    OPTIONAL { <%s> skos:broaderOf ?broaderOf } .
    OPTIONAL { <%s> rdf:subClassOf ?subClassOf } .
    OPTIONAL { <%s> dct:subject ?subject } .
}
```

Figure 11: SPARQL query for finding related resources.

3.3.2. Ontology

The Ontology module provides the necessary functions needed to handle all actions regarding DBpedia. The first step is to find all the mentions of DBpedia sources in the document received from the API module. To accomplish this task, DBpedia Spotlight, described in Section 2.2.1, is employed by simply sending a JSON request to the service with the text that is to be annotated along with the desired confidence and support values (see Section 2.2.1). An example query can be seen in Figure 10. The confidence and support used in this thesis were 0.5 and 20 respectively. DBpedia Spotlight will then answer the request by sending a list of the annotated DBpedia resources. After receiving the requested annotations they are aggregated in a list that contains each unique annotation along with the number of times it appears in the document.

Following Hulpus et al.'s work [6] the n most frequent DBpedia resources are selected as *seed concepts*. DBpedia is then queried using the SPARQL query in Figure 11 to find related DBpedia resources:

In the SPARQL query of Figure 11 all properties of type `skos:broader`, `skos:broaderOf`, `rdf:subClassOf`, and `dct:subject` are selected from a given resource. The `%s` symbol in the query is Python syntax for text formatting. The `%s` is replaced by a resource that is going to be queried against.

This query finds the related DBpedia resources that are connected to the seed concepts with one of the following edge types: `skos:broader`, `skos:broaderOf`, `rdf:subClassOf`, `dct:subject`. The same query is then used to find the related resources of the related resources of the seed concepts. One of these queries is called a hop. It was empirically discovered by Hulpus et al. [6]

that two hops is ideal and thusly in this work, only two hops are used. The found nodes and their relationships to other nodes form a list of edges, where nodes are the DBpedia resources and edges are the connections between those resources. This list of edges is passed to the Graph builder module for further processing.

3.3.3. Graph Builder

The simplest of the modules is the Graph Builder. It has only two tasks: creating the graph from edges provided by the Ontology module, and ranking the nodes in the graph using the degree centrality measure. The nodes represent the topic candidates and the edges the connections between the topics.

The decision to use degree centrality instead of focused information centrality or focused random walk centrality proposed by Hulpus and Greene in [6] stems from research conducted by Boudin in [94]. Although, the Bouding does not compare degree centrality to the measures proposed by Hulpus and Greene, the Bouding concludes that "degree centrality, despite being conceptually the simplest measure, achieves results comparable to the widely used TextRank algorithm".

After the graph is created and the nodes ranked, the nodes are passed to the Ranker module for final ranking.

3.3.4. Ranker

The Ranker module devises the final ranking of the topic candidates. It does it by calculating a *topic overlap score* (tos) for each topic and combining them with the degree centrality scores of each topic given by the Graph builder module. Topic overlap score is calculated with tokenizing the topic and calculating how many times each token in the topic appears in the document and dividing the calculated number by the length of the topic effectively normalizing the length of the topic. The formula of the topic overlap score is as seen in Equation 11:

$$\mathrm{tos} = \frac{\mathrm{tf}}{|\mathrm{t}|}, \tag{11}$$

where tf is the token frequency, or the number of times the token appears in the document, and $|\mathrm{t}|$ is the length of the topic in tokens.

The final topic score (fts) is then combined with the degree centrality score using the following formula:

$$\mathrm{fts} = \alpha \mathrm{dcs} + \beta \mathrm{tos}, \tag{12}$$

where α and β are coefficients for adjusting the weight of the degree centrality score (dcs) and tos respectively. Following the intuition that topics directly mentioned in the document are more important than latent ones, the coefficient β is given a value of 1.5 and α 1.0.

After the final score is calculated the topic candidates are ranked and n highest scoring candidates are selected as the topics of the document.

3.4. Implementation

This section describes how the system was implemented to match the requirements in Section 3.1 and the system design presented in Section 3.3.

3.4.1. Tools and Libraries

The programming language chosen for this project was Python (version 3.6). As noted in Section 3.2, Python is a highly modular language with a large collection of standard libraries. Scientific computing community has been using Python for a long time building a large quantity of libraries suitable for NLP. Python is also very expressive and rather easy to learn, enabling easy collaboration with other programmers in the future of a project.

The libraries used in this work can be found in Table 2.

Table 2: Project libraries

Library	Usage	Description
spaCy [95]	NLP	Provides a large set of functions for NLP
aiosparql [96]	SPARQL	Provides an easy way to integrate SPARQL queries in python
NetworkX [97]	Graphs	Provides functions for building and analysing network graphs
Flask [98]	API	Provides a set of utilities for building Web APIs

- **spaCy** is an "Industrial-Strength Natural Language Processing library" [95]. The authors of the library claim it is the fastest NLP library in the world, in terms of computing time. spaCy provides a simple and unified API for its functions making it very easy to use. These attributes make up for a strong case for using spaCy rather than the more conventional NLTK library. In this thesis, spaCy is used for text tokenization.

- **aiosparql** is an asynchronous SPARQL wrapper for Python [96]. It provides an easy way to integrate SPARQL queries to Python code. There are other SPARQL wrappers for Python but aiosparql is the only one providing asynchronous network calls making network call optimization effortless.

- **NetworkX** is a "Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks" [97]. It is used in this project for building the graphs and analyzing the centrality of the nodes in the graphs.

- **Flask** is a micro-framework for Python for building web APIs [98]. This minimalist framework is very modular enabling the use of only those functions needed thus making APIs build with Flask light and simple.

3.4.2. Internal Modules

The Python code in this work is split into modules, each providing a set of functions that belong to a block seen in Figure 6. The design of the modules can be found in Section 3.3 and a data flow diagram for the same modules can be seen in Figure 12. The colored blocks of Figure 12 are the internal modules of the Topic Distiller. The white circles represent the main functions of the said modules and the arrows represent the data flow between the functions. The sizes and colors of the blocks and circles do not carry any specific meaning.

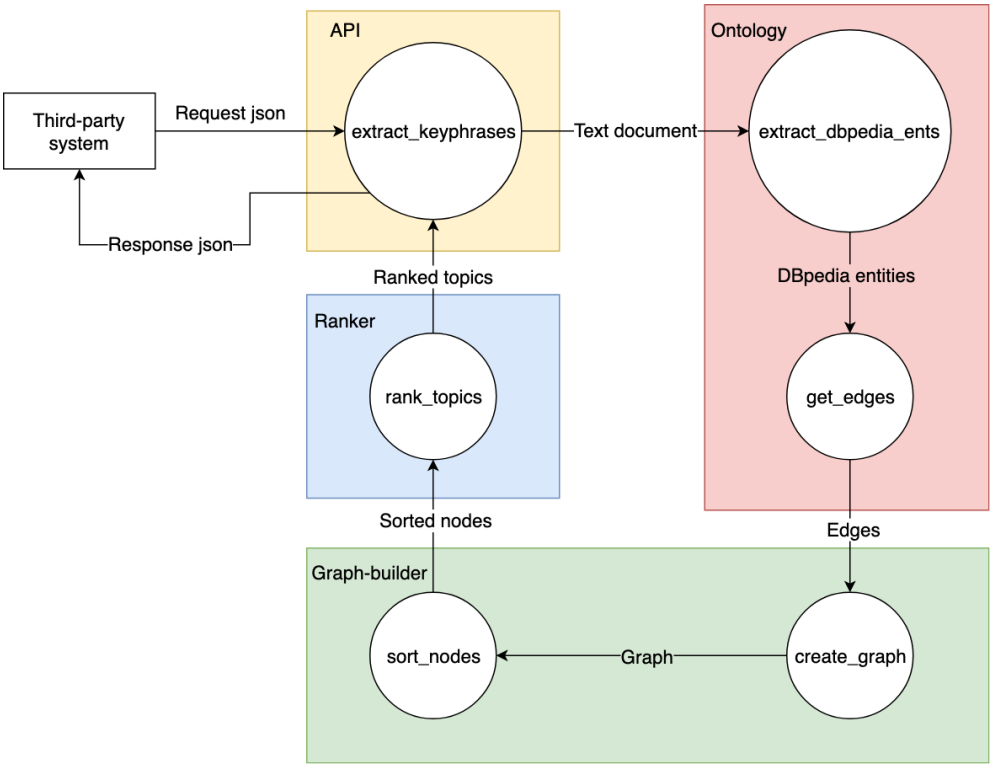


Figure 12: Internal modules data flow diagram.

API

This module (the yellow box top-left in Figure 12) simply exposes the `extract_topics` function to outside world with HTTP protocol using the *Flask* micro-framework. The `extract_topics` function takes only two arguments: the text to be analyzed and the number of topics to be returned. Detailed explanation of the design of the API can be found in Section 3.3.1.

Ontology

This module (the red box top-right in Figure 12) connects the rest of the system to DBpedia. It provides the SPARQL commands needed to traverse DBpedia along with the logic to do so. The main functions in the Ontology module are `extract_dbpedia_ents`, which queries DBpedia Spotlight for annotating

mentions of DBpedia entities in the text, and `get_edges`, which creates a list of linked DBPedia entities by recursively querying DBpedia given the entities annotated by `extract_dbpedia_ents`. The implementation of the Ontology module relies on the *aiosparql* library.

Graph Builder

The Graph Builder module (the green box bottom in Figure 12) comprises of functions that are used to build and analyse the network graph from the list of connected edges provided by the `get_edges` function of the Ontology module. The main functions of the Graph Module are `create_graph`, which takes in a list of edges and builds a graph from them, and `sort_nodes` which sorts the nodes by their degree centrality. The implementation of the Graph Builder module utilises the *NetworkX* module.

Ranker

The Ranker module (the blue box left-middle in Figure 12) takes the sorted graph from Graph Builder and scores and ranks the nodes according to the *topic overlap score* (tos) defined in Equation 12. The Ranking module consists of two main functions: `get_topic_overlap_score`, and `rank_topics`. The former is used to calculate tos for each node or topic and latter for ranking the topics using the *Final Topic Score* (fts) defined in Equation 12. The tokenization needed for computing tos is provided by the *spaCy* library.

4. EVALUATION

The scope of this evaluation was to check whether the implemented described in Chapter 3 satisfies the requirements listed in Section 3.1. The evaluation described in this chapter provides answers to the questions listed further below.

The first and most important question: *Is the Topic Distiller able to identify relevant and latent topics in a textual document?* To answer this, the Topic Distiller is evaluated using the conventional methods discussed in Section 2.3.3. These methods include precision (p), recall (r), F-score (f), and R-precision ($\mathrm{p_R}$). The aforementioned metrics are used to compare the performance of Topic Distiller against a set of state-of-the-art AKE-algorithms listed in Section 4.1.

The second question to arise is: *Is the Topic Distiller able to extract topics from social media postings?* This question is more problematic since no available dataset for social media postings with human annotated keyphrases were found. In order to evaluate the performance of the Topic Distiller on social media postings a dataset of 70 human annotated tweets was created (see Section: 4.1.1).

The third question is: *Do the topics discovered by the Topic Distiller represent real world phenomena*? Fortunately, the usage of EL guarantees this and no evaluation of this requirement is needed.

Performance testing was conducted to answer the question number 4: *Is the Topic Distiller able to process documents fast enough?* The only performance metric of interest was the throughput of the system, or how many documents the system is able to process in a time period.

The fifth question would be: *Is the Topic Distiller easy to maintain?* This question is hard to answer by evaluation. Instead, a number of design principles, described in Section 3.2, were adopted to make the system as maintainable as possible.

The question number six is: *Does the Topic Distiller provide a simple interface for communicating with other systems?* No evaluation was conducted on the simplicity of the API provided by the Topic Distiller, but it is easy to see that the API is simple enough by looking at its design in Section 3.3.1.

Additionally, a seventh question is added to the evaluation: *Does machine translation affect the performance of the Topic Distiller?* This is an important question since the Topic Distiller, in its current form, is only able to process documents written in English. This clearly poses a problem to processing documents written in other languages. One possible way to handle non-English documents is to first translate them to English and them feed them to the Topic Distiller. While machine translation keeps improving from year to year it still is not perfect, making the machine translated document somewhat different from a human translated one.

To evaluate the effects of machine translation on the performance of the Topic Distiller a special dataset, described in Section 4.1.1, is collected. This dataset includes articles written both in Portugese and English. The Portuguese articles are then translated to English using Google Translate service[1]. Then the performance of the Topic Distiller is measured using original (English) and the translated versions of the articles and the results compared.

To summarize, here are the questions to be addressed in this evaluation:

[1] `https://translate.google.com/`

1. Is the Topic Distiller able to identify relevant and latent topics in a textual document?
2. Is the Topic Distiller able to extract topics from short texts such as social media postings?
3. Is the Topic Distiller able to process enough documents fast enough?
4. Does machine translation have an effect to the output of the Topic Distiller?

4.1. Test Setup

This section describes the evaluation setup including the methods and data used.

4.1.1. Datasets

Five datasets were used to evaluate the Topic Distiller. Two of the datasets (hulth2003 and semeval2010) are from the literature. The other three datasets (guardian, tweets, wikinews) are composed for the purposes of this thesis. More information on the composition of these three datasets can be found in the appendix (see Section 8). A summary of datasets can be found in table 3 and descriptions of said datasets are presented further in this section.

The *Type* column of Table 3a describes what sort of document the dataset consists of and the *# of documents* indicates how many documents each datasets has. The *Avg KPD* column tells the average number of keyphrases per document and the *Max KPD* and *Min KPD* columns of 3b indicate what are the maximum and minimum number of keyphrases per document respectively. Similarly the *Avg length*, *Max length*, and *Min length* columns of Table 3c express the average, maximum, and minimum length of documents, measured by characters, found in a dataset. The *% in Wikipedia* column of Table 3b depicts the number of keyphrases that are also Wikipedia article titles[2]. This information is relevant since the Topic Distiller is only able to find topics that are also Wikipedia article titles. The *% of latent* column of Table 3c shows which percentage of keyphrases in the dataset are latent in their respective documents.

1. **Hulth2003**
   The Hulth2003 (hulth2003) [43] is a dataset of 2000 scientific article abstracts in English. The abstracts were collected from journal papers during the years 1998-2002 and the disciplines include *Computers and Control*, and *Information Technology*. The 2000 articles are randomly split into three sets: *training*, *testing*, and *validation*, each consisting of 1000, 500, and 500 abstracts respectively. We used the 500 abstracts from validation dataset. The maximum number of keyphrases per document in the validation set is 28 and minimum 1. The longest document is 1843 characters long, while the shortest is only 105 characters. Latent keyphrases are at 23.1% of all the keyphrases in this dataset.

[2]The set of Wikipedia article titles was downloaded from https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-all-titles.gz on 22nd of January 2019

Table 3: Dataset descriptions

(a)

Name	Type	# of documents
hulth2003	Abstracts	500
semeval2010	Scientific articles	144
guardian	News articles	190
tweets	Social media postings	60
wikinews	News articles	135

(b)

Name	Avg KPD	Max KPD	Min KPD	% in Wikipedia
hulth2003	9.15	28	1	13.3
semeval2010	3.88	13	2	26.7
guardian	5.75	19	2	48.4
tweets	4.33	9	1	46.5
wikinews	9.13	26	3	38.9

(c)

Name	Avg length	Max length	Min length	% of latent
hulth2003	805	1843	105	23.1
semeval2010	48100	86483	16707	25.9
guardian	4070	39334	964	52.8
tweets	137	288	43	44.6
wikinews	4740	18163	2110	42.4

2. **SemEval-2010 Task 5**
   The SemEval-2010 Task 5 (semeval2010) [99] dataset consists of conference and workshop papers totaling 284 documents. The documents fall into the following disciplines: *Distributed Systems*, *Information Search and Retrieval*, *Distributed Artificial Intelligence – Multiagent Systems*, and *Social and Behavioral Sciences – Economics*.

   In contrast to hulth2003 dataset, this datasets contains the whole articles, including the abstract. The data is split into *trial*, *train*, and *test* sets with 40, 144, and 100 articles. The trial data is a subset of the training data which decreases the amount of unique documents to 244 documents. In this work, the train dataset is used. Most keyphrases per document in the train set is 13 and the least keyphrases per document is 2. 86483 is the length of the longest document, measured in characters, the shortest being 16707. Latent keyphrases make 25.9% of the total keyphrases.

3. **Guardian**
   The Guardian (guardian) dataset is the first of the three datasets created specifically for this work. The dataset consists of 200 news articles collected from The Guardian[3] web news site during August of 2018. All of the articles include annotations for topics present in the article as can be seen from Figure

[3] `https://theguardian.com`

13. 19 is the maximum number of keyphrases per document and 2 is the minimum. The longest document is 39334 characters long and the shortest lies only at 964 characters. The percentage of latent keyphrases of total keyhprases is 52.8.

    In addition to consisting of news articles instead of scientific articles or abstracts, the keyphrases in the guardian dataset are often absent from the documents themselves. This fact makes the guardian dataset a good candidate for evaluating the performance of AKE algorithms on latent keyphrases.

    Unfortunately, due to copyright issues, this dataset cannot be distributed.

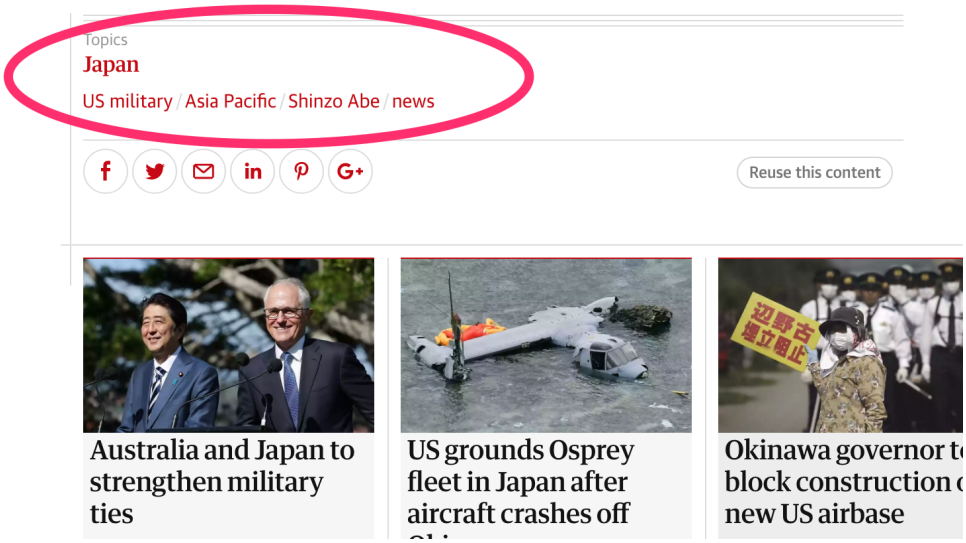


Figure 13: The Guardian article topics source: `https://www.theguardian.com/world/2018/oct/01/japan-us-military-base-critic-voted-in-as-okinawa-governor`.

4. **Tweets**
   The Tweets (tweets) dataset is also created for this evaluation. It consists of 60 randomly selected tweets collected on 20th of September of 2018. The tweets were annotated by the author of this work. The annotations include keyphrases that are present in the tweets themselves and some keyphrases that cannot be found in the documents. Most keyphrases per document resides at 9 and least at 1. The longest tweet is 288 characters in length while the shortest is 43 characters long. 44.6% of all keyphrases in this dataset are latent.

   The distribution of this dataset is also prohibited by the copyright issued by Twitter.

5. **Wikinews**
   Like the guardian and tweets datasets, the wikinews dataset is also composed for the purpose of evaluating the Topic Distiller. It consists of 135 articles collected from wikinews[4]. What sets this particular dataset apart from the others, is the fact that this dataset consists of sets of news articles both in Portuguese and

[4] `https://en.wikinews.org/`

English. The purpose of this dataset is to enable evaluation of the effect of machine translation on different metrics of evaluation. The test setup, which employs this dataset, is detailed in Section 4.4. Maximum number of human annotated keyphrases in a document lies at 26 and minimum at 3, while lengths of the document reside between the minimum of 2110 characters and maximum of 18163 characters. The percentage of latent keyphrases of all keyphrases is 42.4.

4.1.2. Data Preparation

The only preprocessing done to the documents and their keyphrases is lowercasing. No further preprocessing is done to emulate the messy reality where it is difficult to know what kind of textual data might be fed to the system.

4.1.3. Latent Keyphrases

Evaluating the ability of AKE methods to discover latent keyphrases, or keyphrases that are not to be found in the documents themselves, needs special consideration. For example, should a keyphrase be labeled as present if parts of it are present in the document? Or should the keyphrase be thought as latent if it is constructed from words all found within the document but the keyphrase itself is not in the document?

In this work, a keyphrase is classified as present if all of its component words are found within the document and latent if one ore more words are missing in the document. Non alphanumerics are removed from the keyphrases before evaluating if they are latent or not.

To evaluate the performance of AKE methods on discovering latent keyphrases a modified versions of the datasets, of section 4.1.1, are created. In these modified datasets the present keyphrases are removed leaving only the latent ones.

4.2. Metrics

This section details the metrics used to evaluate the Topic Distiller.

4.2.1. Precision, Recall, F-score, and R-precision

To evaluate the performance of Topic Distiller versus state-of-the-art models we used the metrics p, r, f, and $\mathrm{p_R}$, defined in Section 2.3.3 by equations 5, 6, 7, 10 respectively.

The metrics, p, r, and f, are computed first with 5 extracted keyphrases and then with 10 and 15 extracted keyphrases.

The metric $\mathrm{p_R}$ is computed three times per dataset using the three different strategies, *Exact*, *Includes*, and *PartOf*, proposed by [77]. The Exact strategy compares keyphrases as they are; only *exact* matches are counted as correct ones. The Includes method also considers extracted keyphrases that include a standard one as a

substring as correct. For example, an extracted keyphrase *neural machine translation* would be considered as correct if the corresponding standard keyphrase is *machine translation*. Conversely, the PartOf method considers an extracted keyphrase as correct if it is a part of an standard keyphrase. For example, an extacted keyphrase *artifical intelligence* is considered correct if the standard is *general artifical intelligence*. For each strategy, lemmatization is used to reduce the words in the keyphrases into their base forms. For precision, recall, and f-score no such lemmatization is used. To make the lengths of the lists of extracted and standard keyphrases equal in length 15 keyphrases are extracted using each method and the length of the longer list of keyphrases is truncated to match the length of the shorter list. For example, if the length of standard keyphrases is 12 then three keyphrases from the end of the extracted keyphrases are cut off.

In addition to the aforementioned metrics, a modified version of each method, where the ranking is not taken into account, is introduced. Instead correct keyphrases are computed as the intersection of sets of standard and extracted keyphrases. These modified metrics are used to evaluate how much of the latent keyphrases the systems are able to identify.

4.2.2. Processing Time

Two tests are composed for evaluating the processing time of the algorithms. The first test evaluates the effect of document length on the processing time. The test is devised so that a document is truncated to be only 1000 characters of length. Then, 1000 characters are added at each step and the processing time is recorded. At each step the measurement is performed ten times to average out possible fluctuations of time caused by the underlying operating system. The document used in this test is the longest of the documents in the guardian dataset.

The second test evaluates how the number of keyphrases extracted affects the processing time. This is done by incrementing the number of keyphrases from one to 20. Again, at each step the measurement is repeated ten times and the final value is the average of those ten times.

All processing time tests were conducted on an Apple MacBook Pro (15-inch, 2018) laptop with 2,6 GHz Intel Core i7 processor and 16 GB 2400 MHz DDR4 memory. The operating system used was macOS Mojave version 10.14.3.

4.3. State-of-the-art AKE methods

To compare the performance of state-of-the-art AKE methods to the Topic Distiller, we employ the pke – python keyphrase extraction [100] toolkit which contains implementations for multiple state-of-the-art AKE methods. We use the methods described in Table 4.

These methods include only unsupervised graph-based AKE methods. This makes sense since the method proposed in this work is also an unsupervised graph-based method.

Table 4: AKE algorithms used in evaluation

Name	Author(s)	Method
Topical PageRank	Sterckx et al. [61]	Topical information with PageRank
PositionRank	Florescu et al. [54]	Word position with PageRank
TopicRank	Bougouin et al. [49]	Topic clusters with PageRank
MultipartiteRank	Boudin [56]	Multipartite Graphs with PageRank

The author of the pke toolkit, Boudin, evaluates the performance of the algorithms implemented in the toolkit with the semeval2010 test dataset using F-score and 10 top keyphrases without taking ranking of the keyphrases into account. The version of the standard keyphrases Boudin uses in the evaluation are the combined author- and reader-assigned keyphrases. Stemming is also used in both standard and extracted keyphrases. The results of this evaluation can be seen in Table 5.

Table 5: Evaluation of pke algorithms by Boudin fetched from: `https://github.com/boudinfl/pke` in November 2018

Name	F-score
Topical PageRank	0.031
PositionRank	0.067
TopicRank	0.119
MultipartiteRank	0.142

Table 6: Reproduced evaluation of pke algorithms

Name	F-score
Topical PageRank	0.029
PositionRank	0.065
TopicRank	0.118
MultipartiteRank	0.138

A reproduction of Boudin's evaluation was conducted to make sure that the results Boudin reports are valid and that the pke toolkit is safe to use in the evaluation of the Topic Distiller. Results of this reproduction are found in Table 6. By comparing the F-scores, reported by Boudin (see Table 5), to the reproduced F-scores (see Table 6), it can be noted that the use of pke is safe, since the scores differ only marginally.

4.4. Effect of Machine Translation

The effect of machine translation on the performance of the Topic Distiller was also evaluated. This was done by measuring precision, recall, F-score, and R-score with two different sets of documents. The first set are English news articles and the second set are the same articles written in Portuguese and then translated to English using the Google Translate[5] machine translation service. Then the aforementioned metrics are

[5]https://translate.google.com/

computed using both the original English version and the translated English version in order to see the impact of machine translation on this system.

This evaluation is carried out because, as it is described in this thesis, the Topic Distiller is only able to extract keyphrases from documents written in English. Machine translation is one possible solution to extract keyphrases from documents written in other languages.

4.5. Results

This section lists the results of the evaluation described in Section 4.1.

4.5.1. Precision, Recall, and F-score

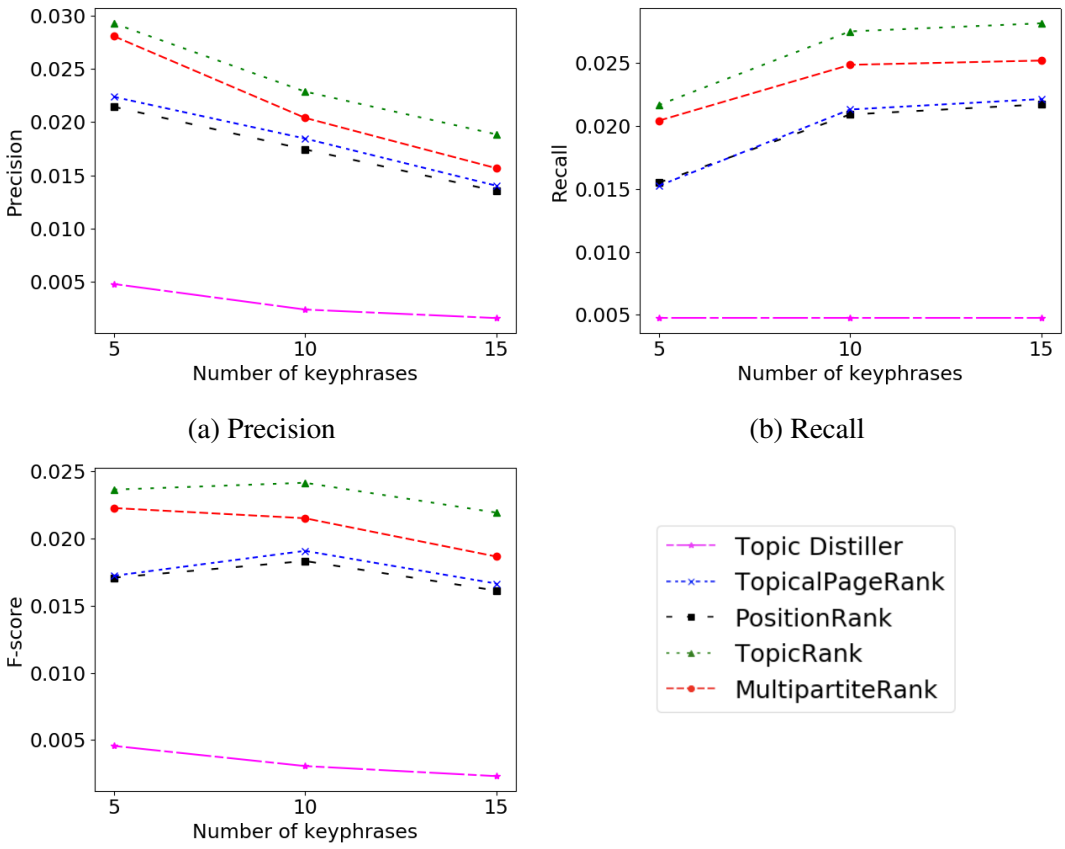
This section lists the results of p, r, and f measurements. The results are grouped so that each dataset has its own page where the evaluation results are presented along with a short explanation. The highest score in each column is in bold and the method that has the highest mean score is in bold also. Note that the results of measurements with 10 keyphrases are omitted from the tables for brevity but are still visible on the plot figures.

hulth2003

The results depicted in Table 7 and Figure 14 show that the Topic Distiller is not optimal for extracting keyphrases from abstracts of scientific articles. Far better results can be achieved using conventional methods from which TopicRank has the top performance. The poor performance of Topic Distiller could be attributed to two factors: 1. Many of the keyphrases in the hulth2003 dataset (86.7%, see Table 3b) can not be found in the wikipedia as article titles. 2. The keyphrases in the hulth2003 dataset are very specific and the Topic Distiller is biased towards more general concepts.

Table 7: p, r, and f evaluated on hulth2003 dataset with 5 and 15 keyphrases

Method	p@5	p@15	r@5	r@15	f@5	f@15
TD [*]	0.0048	0.0016	0.0047	0.0047	0.0046	0.0023
TPR [61]	0.0224	0.014	0.0152	0.0221	0.0172	0.0166
PR [54]	0.0214	0.0135	0.0155	0.0217	0.0171	0.0161
TR [49]	**0.0293**	**0.0188**	**0.0217**	**0.0281**	**0.0236**	**0.0219**
MR [56]	0.0281	0.0157	0.0204	0.0252	0.0223	0.0187



(a) Precision (b) Recall

(c) F-score (d) Legend

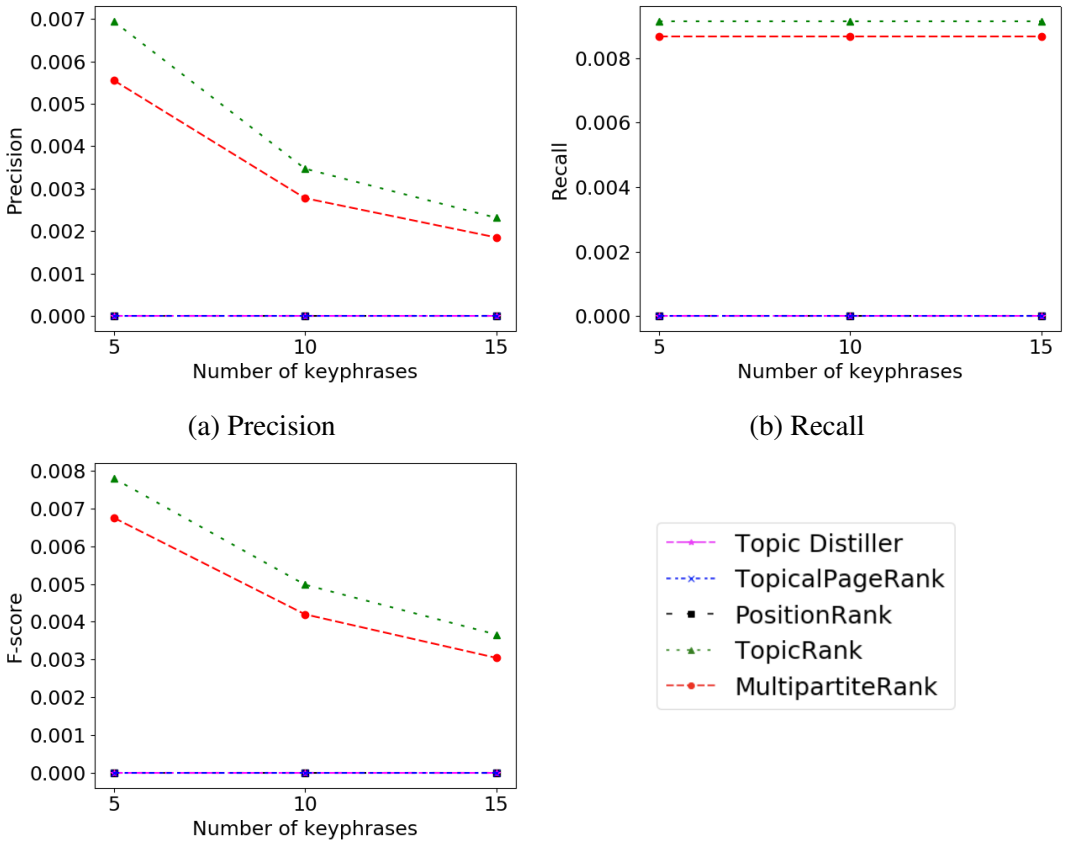
Figure 14: Precision, recall, and F-score on hulth2003 dataset.

semeval2010

The evaluation results on the semeval2010 dataset seen in Table 8 and Figure 15 show that none of the algorithms compared perform very well on this dataset. Only TopicRank and MultipartiteRank are able to extract any matches and even their performance is an order of magnitude worse than with the hulth2003 dataset. This is due to the fact that, unlike the hulth2003 dataset, semeval2010 dataset consists of whole scientific articles with non-human language, like mathematical notation, included, making the extraction process much harder.

Table 8: p, r, and f evaluated on semeval2010 dataset with 5 and 15 keyphrases

Method	p@5	p@15	r@5	r@15	f@5	f@15
TD [*]	0.0	0.0	0.0	0.0	0.0	0.0
TPR [61]	0.0	0.0	0.0	0.0	0.0	0.0
PR [54]	0.0	0.0	0.0	0.0	0.0	0.0
TR [49]	**0.0069**	**0.0023**	**0.0091**	**0.0091**	**0.0078**	**0.0037**
MR [56]	0.0056	0.0019	0.0087	0.0087	0.0068	0.003



(a) Precision (b) Recall

(c) F-score (d) Legend

Figure 15: Precision, recall, and F-score on semeval2010 dataset.

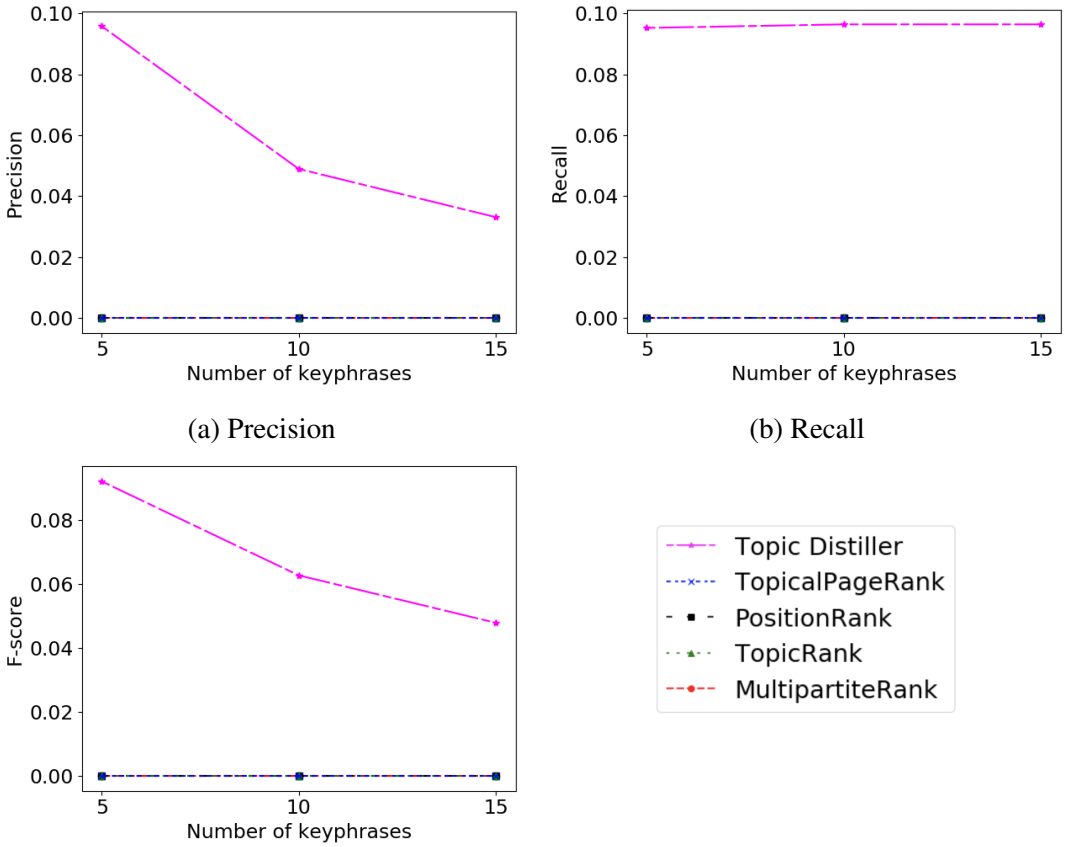
guardian

As it can be noted from looking at Table 9 and Figure 16, Topic Distiller is the only algorithm that scores above zero in precision, recall and F-score when evaluated on the guardian dataset. The reason for this stems from the facts that the guardian dataset contains the highest percentage of latent keyphrases (52.8%, see Table 3c) and that almost half (48.4%, see Table 3b) of the keyphrases are also Wikipedia article titles.

The keyhprases in the guardian dataset are also more general, opposed to being specific, than in the hulth2003 and semeval2010 datasets making Topic Distiller more suitable for finding these keyphrases.

Table 9: p, r, and f evaluated on guardian dataset with 5 and 15 keyphrases

Method	p@5	p@15	r@5	r@15	f@5	f@15
TD [*]	**0.0958**	**0.0331**	**0.0952**	**0.0964**	**0.0922**	**0.048**
TPR [61]	0.0	0.0	0.0	0.0	0.0	0.0
PR [54]	0.0	0.0	0.0	0.0	0.0	0.0
TR [49]	0.0	0.0	0.0	0.0	0.0	0.0
MR [56]	0.0	0.0	0.0	0.0	0.0	0.0



(a) Precision

(b) Recall

(c) F-score

(d) Legend

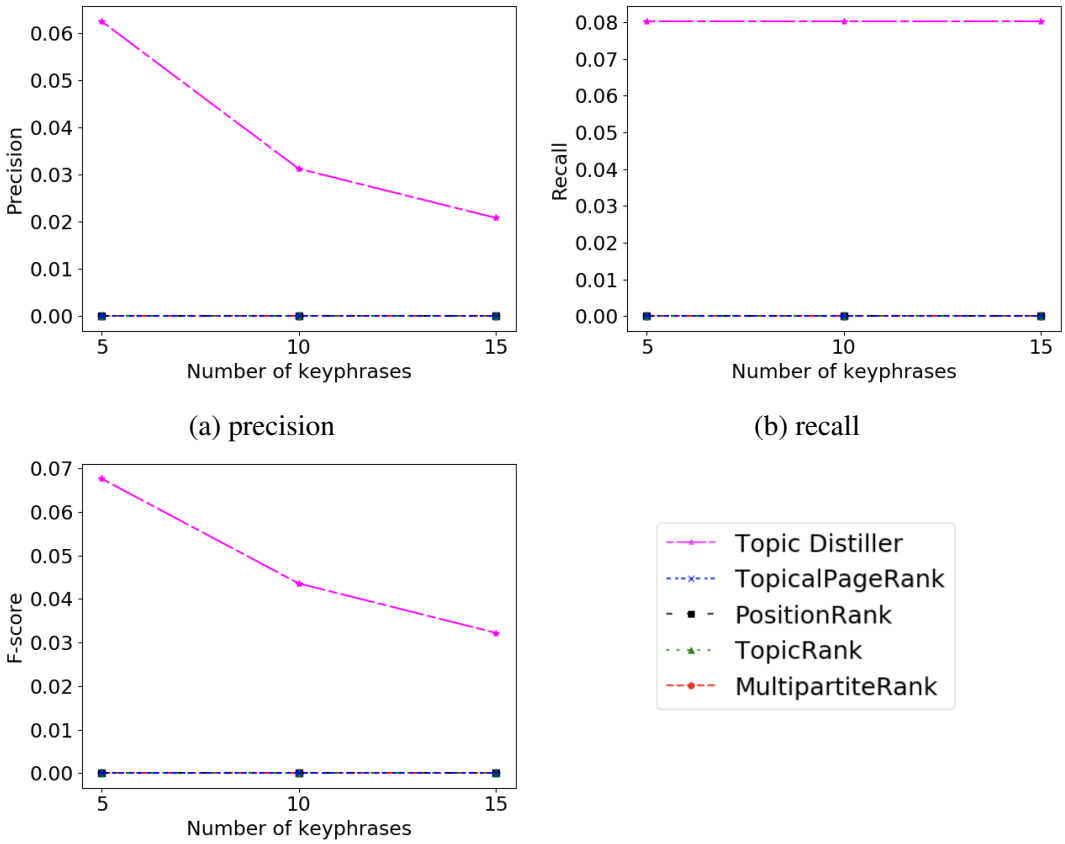
Figure 16: Precision, recall, and F-score on guardian dataset.

tweets

Looking at Table 10 and Figure 17 it is clear that Topic Distiller dominates in performance when tweets are in question. This can be attributed to the fact that tweets are very short texts. Also the tweets dataset contains high percentage of latent keyphrases (44.6%, see Table 3c) and a large portion of the keyphrases are also Wikipedia article titles (46.5%, see Table 3b).

Table 10: p, r, and f evaluated on tweets dataset with 5 and 15 keyphrases

Method	p@5	p@15	r@5	r@15	f@5	f@15
TD [*]	**0.0625**	**0.0208**	**0.0803**	**0.0803**	**0.0676**	**0.0322**
TPR [61]	0.0	0.0	0.0	0.0	0.0	0.0
PR [54]	0.0	0.0	0.0	0.0	0.0	0.0
TR [49]	0.0	0.0	0.0	0.0	0.0	0.0
MR [56]	0.0	0.0	0.0	0.0	0.0	0.0



(a) precision (b) recall

(c) f-score (d) Legend

Figure 17: precision, recall, and f-score on tweets dataset.

4.5.2. R-precision

This section contains the results of $\mathrm{p_R}$ measurements described in Section:4.2.1. The Tables 11, 12, 13, and 14 presenting the $\mathrm{p_R}$ measurements clearly show that $\mathrm{p_R}$ is more forgiving metric than p, r, f (see Tables 7, 8, 9, 10) are. This was expected since $\mathrm{p_R}$ allows for more leeway when the automatically discovered keyphrases are matched against the gold standard ones, as detailed in Section 4.2.

hulth2003

As it was the case with p, r, and f, the TopicRank (TR) performs the best on scientific abstracts when measured with R-precision as seen in Table 11. MultipartiteRank (MR) comes as close second scoring best when measured with *PartOf* strategy. The Topic Distiller (TD) has the worst performance of the methods but achieves scores comparable to the top methods when measured with the *PartOf* strategy.

Table 11: R-precision evaluated on hulth2003 dataset

Method	Exact	PartOf	Includes
TD [*]	0.0183	0.0543	0.0231
TPR [61]	0.0262	0.0532	0.0392
PR [54]	0.0256	0.0549	0.0343
TR [49]	**0.0342**	0.0561	**0.0411**
MR [56]	0.0301	**0.0574**	0.0357

semeval2010

R-precision measured on the semeval2010 dataset, seen in Table 12, shows interesting results as the top performer is not TopicRank (TR) as usual. Here, the Topic Distiller (TD) scores highest points when measured with the *Exact* strategy, MultipartiteRank (MR) achieves best performance with the *PartOf* strategy, and PositionRank (PR) with the *Includes* strategy.

Table 12: R-precision evaluated on semeval2010 dataset

Method	Exact	PartOf	Includes
TD [*]	**0.0149**	0.025	0.0206
TPR [61]	0.0	0.0014	0.0406
PR [54]	0.0051	0.0086	**0.0471**
TR [49]	0.0105	0.0417	0.0156
MR [56]	0.0122	**0.0538**	0.0235

guardian

By looking at the Table 13, it is evident that the Topic Distiller achieves by far the best scores in the guardian dataset. Worth mentioning is the fact that all the other methods

scored zero points on this dataset when measured with the conventional precision, recall, and F-score. This combined with the fact that relative change in performance measured with the three different strategies – especially the difference between the *Exact* strategy and the other two strategies – is not very large indicates that the list of keyphrases extracted and the list of standard keyphrases in the dataset are of different lengths.

Table 13: R-precision evaluated on guardian dataset

Method	Exact	PartOf	Includes
TD [*]	**0.0964**	**0.1031**	**0.1199**
TPR [61]	0.009	0.0134	0.0282
PR [54]	0.0092	0.0184	0.0305
TR [49]	0.0147	0.0188	0.0202
MR [56]	0.0118	0.0154	0.0168

tweets

The Table 14 shows that, again, the Topic Distiller dominates when tweets are in question, but the gap between performance of Topic Distiller to other methods is not as large as in the guardian dataset.

Table 14: R-precision evaluated on tweets dataset

Method	Exact	PartOf	Includes
TD [*]	**0.0881**	**0.0946**	**0.1245**
TPR [61]	0.0413	0.0484	0.0849
PR [54]	0.0543	0.0575	0.0865
TR [49]	0.0621	0.0673	0.0972
MR [56]	0.0601	0.0653	0.0953

4.5.3. Latent Keyphrases

The Table 15 clearly shows that the Topic Distiller (TR) is the only method present in the comparison able to find latent keyphrases. The semeval2010 dataset is evidently the hardest dataset to extract keyhprases from. None of the methods was able to find any of the latent keyphrases from it. The Topic Distiller performs best on the tweets dataset by a large margin. Worth noting is that the keyphrases of the tweets dataset were annotated by the author of this thesis, so there might be a bias present skewing the results.

Table 15: Precision (p), recall (r), and F-score (f) evaluated on latent keyphrases with 5 and 15 keyphrases

(a) hulth2003

Method	p@5	p@15	r@5	r@15	f@5	f@15
TD [*]	**0.0124**	**0.0193**	**0.0082**	**0.0056**	**0.0073**	**0.0034**
TPR [61]	0.0	0.0	0.0	0.0	0.0	0.0
PR [54]	0.0	0.0	0.0	0.0	0.0	0.0
TR [49]	0.0	0.0	0.0	0.0	0.0	0.0
MR [56]	0.0	0.0	0.0	0.0	0.0	0.0

(b) semeval2010

Method	p@5	p@15	r@5	r@15	f@5	f@15
TD [*]	0.0	0.0	0.0	0.0	0.0	0.0
TPR [61]	0.0	0.0	0.0	0.0	0.0	0.0
PR [54]	0.0	0.0	0.0	0.0	0.0	0.0
TR [49]	0.0	0.0	0.0	0.0	0.0	0.0
MR [56]	0.0	0.0	0.0	0.0	0.0	0.0

(c) guardian

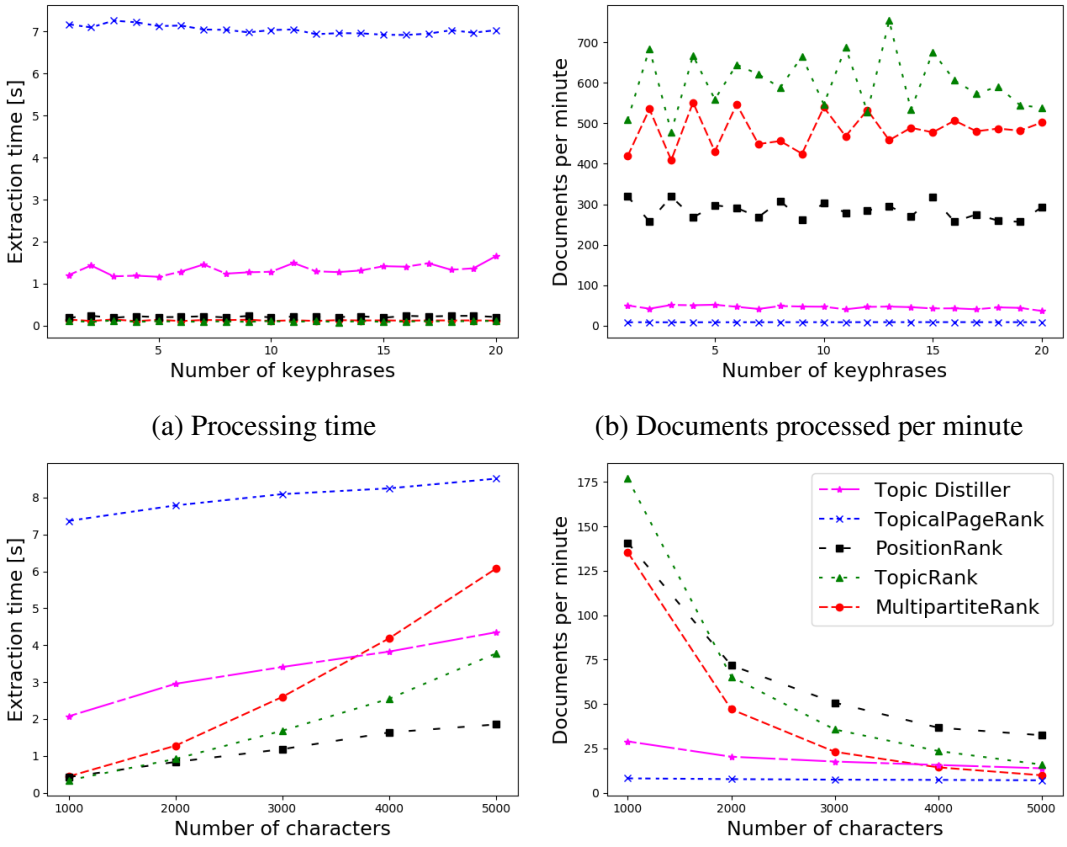
Method	p@5	p@15	r@5	r@15	f@5	f@15
TD [*]	**0.0066**	**0.0151**	**0.0059**	**0.0055**	**0.0053**	**0.0035**
TPR [61]	0.0	0.0	0.0	0.0	0.0	0.0
PR [54]	0.0	0.0	0.0	0.0	0.0	0.0
TR [49]	0.0	0.0	0.0	0.0	0.0	0.0
MR [56]	0.0	0.0	0.0	0.0	0.0	0.0

(d) tweets

Method	p@5	p@15	r@5	r@15	f@5	f@15
TD [*]	**0.2**	**0.2203**	**0.1275**	**0.0596**	**0.1037**	**0.0358**
TPR [61]	0.0	0.0	0.0	0.0	0.0	0.0
PR [54]	0.0	0.0	0.0	0.0	0.0	0.0
TR [49]	0.0	0.0	0.0	0.0	0.0	0.0
MR [56]	0.0	0.0	0.0	0.0	0.0	0.0

4.5.4. Processing Time Performance

This section covers the evaluation results on the processing time performance.



(a) Processing time

(b) Documents processed per minute

(c) Processing time

(d) Documents processed per minute

Figure 18: Extraction time and documents processed per minute as functions of number of extracted keyphrases and document length in characters.

As it can be seen from Figure 18 and Tables 16, 17, 18, and 19, TopicRank (TR) and PositionRank (PR) are the fastest of the methods in this comparison. Slowest are the Topic Distiller (TD) and TopicalPagerank (TPR). That said, all algorithms are able to extract keyphrases in a reasonably short amount of time, that is, they work at a throughput that can not be reasonably expected from any individual human being.

It is also evident that the number of extracted keyphrases does not have an impact on the processing time in any of the methods. This is due to the fact that the number of keyphrases extracted is set by taking the top n keyphrases from a set of keyphrase candidates. Selecting the top 5 candidates takes no longer than choosing 15 or 20 of the top candidates.

Document length, on the other hand, is inversely proportional to the processing time. This is expected, since more text means more data to analyze. MultipartiteRank (MR) is most affected by the length of the documents while TD does not take a big hit on the processing time when the length of the document increases.

It should be noted that very fluctuating behavior of TR, MR, and PR seen in Figure: 18b is due to the fact that these algorithms are so fast that slight changes

in processing speeds, caused by the underlying operating system, result in drastic difference in the amount of documents processed, even when the results are an average of multiple measurements.

Table 16: Processing time in seconds with extracted keyphrases

Num of keyphrases	TD [*]	TPR [61]	PR [54]	**TR [49]**	MR [56]
1	1.2030	7.1724	0.1879	0.1180	0.1432
2	1.4323	7.0978	0.2332	0.0876	0.1121
3	1.1736	7.2587	0.1881	0.1254	0.1467
4	1.1896	7.2197	0.2243	0.0899	0.1088
5	1.1618	7.1299	0.2016	0.1073	0.1394
6	1.2874	7.1423	0.2063	0.0930	0.1097
7	1.4561	7.0473	0.2235	0.0966	0.1337
8	1.2373	7.0408	0.1947	0.1020	0.1315
9	1.2705	6.9800	0.2302	0.0902	0.1412
10	1.2821	7.0352	0.1979	0.1097	0.1112
11	1.4904	7.0497	0.2149	0.0872	0.1282
12	1.2925	6.9375	0.2109	0.1136	0.1126
13	1.2740	6.9614	0.2031	0.0796	0.1309
14	1.3134	6.9572	0.2226	0.1125	0.1227
15	1.4151	6.9232	0.1893	0.0887	0.1256
16	1.4029	6.9182	0.2335	0.0990	0.1184
17	1.4851	6.9498	0.2183	0.1047	0.1249
18	1.3288	7.0306	0.2312	0.1015	0.1232
19	1.3654	6.9722	0.2339	0.1101	0.1245
20	1.6553	7.0326	0.2051	0.1114	0.1195
mean	1.3358	7.0428	0.2125	**0.1014**	0.1254

Table 17: Documents processed per minute with extracted keyphrases

Num of keyphrases	TD [*]	TPR [61]	PR [54]	**TR [49]**	MR [56]
1	49.8753	8.3654	319.3188	508.4746	418.9944
2	41.8907	8.4533	257.2899	684.9315	535.2364
3	51.1247	8.2659	318.9793	478.4689	408.9980
4	50.4371	8.3106	267.4989	667.4082	551.4706
5	51.6440	8.4153	297.6190	559.1799	430.4161
6	46.6056	8.4007	290.8386	645.1613	546.9462
7	41.2060	8.5139	268.4564	621.1180	448.7659
8	48.4927	8.5218	308.1664	588.2353	456.2738
9	47.2255	8.5960	260.6429	665.1885	424.9292
10	46.7982	8.5285	303.1834	546.9462	539.5683
11	40.2576	8.5110	279.1996	688.0734	468.0187
12	46.4217	8.6486	284.4950	528.1690	532.8597
13	47.0958	8.6190	295.4210	753.7688	458.3652
14	45.6830	8.6242	269.5418	533.3333	488.9976
15	42.3998	8.6665	316.9572	676.4374	477.7070
16	42.7686	8.6728	256.9593	606.0606	506.7568
17	40.4013	8.6333	274.8511	573.0659	480.3843
18	45.1535	8.5341	259.5156	591.1330	487.0130
19	43.9432	8.6056	256.5199	544.9591	481.9277
20	36.2472	8.5317	292.5402	538.5996	502.0921
mean	45.2836	8.5209	283.8997	**599.9356**	482.2860

Table 18: Processing time in seconds with document length measured in characters

Document length	TD [*]	TPR [61]	**PR [54]**	TR [49]	MR [56]
100	1.1434	6.9130	0.0505	0.0192	0.0227
200	1.3920	6.9871	0.1105	0.0399	0.0418
300	1.4132	7.0379	0.1485	0.0720	0.0835
400	1.7153	7.1155	0.1847	0.1104	0.1134
500	1.2846	7.1482	0.2216	0.1409	0.1678
600	1.3257	7.2345	0.3141	0.1870	0.2104
700	1.4027	7.2623	0.3035	0.2341	0.2614
800	1.8234	7.2903	0.3567	0.2515	0.3200
900	1.8196	7.3307	0.4364	0.2848	0.3734
1000	2.0734	7.3674	0.4264	0.3393	0.4433
2000	2.9537	7.7866	0.8365	0.9179	1.2767
3000	3.4089	8.0927	1.1825	1.6830	2.5987
4000	3.8271	8.2494	1.6362	2.5484	4.1863
5000	4.3499	8.5118	1.8523	3.7777	6.0764
mean	1.7630	7.2350	**0.3655**	0.4675	0.7100

Table 19: Documents processed per minute with document length measured in characters

Document length	TD [*]	TPR [61]	PR [54]	**TR [49]**	MR [56]
100	52.4751	8.6793	1188.1188	3125.0000	2643.1718
200	43.1034	8.5873	542.9864	1503.7594	1435.4067
300	42.4568	8.5253	404.0404	833.3333	718.5629
400	34.9793	8.4323	324.8511	543.4783	529.1005
500	46.7071	8.3937	270.7581	425.8339	357.5685
600	45.2591	8.2936	191.0220	320.8556	285.1711
700	42.7746	8.2618	197.6936	256.3007	229.5333
800	32.9056	8.2301	168.2086	238.5686	187.5000
900	32.9743	8.1848	137.4885	210.6742	160.6856
1000	28.9380	8.1440	140.7129	176.8347	135.3485
2000	20.3135	7.7055	71.7274	65.3666	46.9962
3000	17.6010	7.4141	50.7400	35.6506	23.0885
4000	15.6777	7.2733	36.6703	23.5442	14.3325
5000	13.7934	7.0490	32.3922	15.8827	9.8743
mean	40.9898	8.3249	930.4084	**1911.8406**	1769.8992

4.5.5. The Effect of Translation

This section lists the results of the evaluation of the effect of translation on the different evaluation metrics.

The Tables 20, 21, and 22 clearly show that the translation has a negative effect on the evaluation metrics. Precision takes the worst hit, while R-precision with the PartOf strategy is least affected by the translation.

Table 20: Original vs. translated with 5 keyphrases

Metric	Original	Translated	Relative change [%]
Precision	0.0222	0.0133	-40.0
Recall	0.0117	0.0081	-30.5
F-score	0.015	0.01	-33.5
R-precision Exact	0.013	0.0105	-18.8
R-precision Includes	0.0335	0.0282	-15.9
R-precision PartOf	0.0139	0.0121	-12.6

Table 21: Original vs. translated with 10 keyphrases

Metric	Original	Translated	Relative change [%]
Precision	0.0126	0.0089	-29.4
Recall	0.013	0.0105	-18.8
F-score	0.0124	0.0095	-23.4
R-precision Exact	0.013	0.0105	-18.8
R-precision Includes	0.0335	0.0282	-15.9
R-precision PartOf	0.0139	0.0121	-12.6

Table 22: Original vs. translated with 15 keyphrases

Metric	Original	Translated	Relative change [%]
Precision	0.0094	0.0066	-29.6
Recall	0.013	0.0105	-18.8
F-score	0.0106	0.008	-24.6
R-precision Exact	0.013	0.0105	-18.8
R-precision Includes	0.0335	0.0282	-15.9
R-precision PartOf	0.0139	0.0121	-12.6

5. DISCUSSION

The goal of the evaluation, detailed in Chapter 4, was to check that the requirements, set in Section 3.1, were fulfilled by the Topic Distiller. Additionally, the effect of machine translation to the performance of the Topic Distiller was evaluated.

This chapter discusses the results of that evaluation recalling the design and implementation details of Chapter 3.

5.1. General Discussion

Tables 7 and 8, along with Figures 14 and 15 reveal that the Topic Distiller is not the best candidate for extracting topics from scientific articles. The TopicRank algorithm dominates on performance in this domain with MultipartiteRank as close second. In terms of precision, recall and F-score, the Topic Distillers performs worst of all the candidates but achieves comparable results with R-precision with PartOf method.

Topic Distiller redeems itself by achieving best performance on the semeval2010 dataset when measured with R-precision using the Exact and PartOf strategies, as can be seen from Table 12. However, the Topic Distiller looses to PositionRank when measured with R-precision using the Includes strategy and to TopicRank when precision, recall and F-score are in question, as is evident from Table 8.

By far the best performance of Topic Distiller is achieved with the guardian and tweet datasets, as shown in Tables 9, 10, 13, and 14. Here, the Topic Distillers performance far surpasses any of the comparison algorithms by all metrics, except for R-precision with strategy Includes measured on the tweets dataset, where TopicRank is the best performing method. The ability to find topics from news articles and, especially, from social media postings is very important in the modern world, because of the overwhelming quantity of this type of data freely available on the internet.

As a summary, the evaluation test results of Section 4.5 show that the Topic Distiller is able to identify both relevant and latent topics in textual content. It works best on news articles and social media postings compared to the state-of-the-art methods but it is unable to reach the preformance of those methods when evaluated on abstracts of scientific articles.

5.2. Satisfying the Requirements

This section takes apart the requirements listed in Section 3.1 and checks that the Topic Distiller fulfills them one by one.

5.2.1. Is the Topic Distiller Able to Identify Relevant and Latent Topics in Textual Content?

None of the compared state-of-the-art methods were able to find any of the latent keyphrases from the datasets. Only Topic Distiller was able to do so. This stems from the fact that these methods only *extract* keyphrases from documents. They may

be able to combine the words of the document in novel ways but the vocabulary is strictly restricted to the document processed.

The poor performance of all algorithms on the semeval2010 dataset is most likely do to the fact that large portions of the documents in the datasets are not natural human language.

5.2.2. Is the Topic Distiller Able Extract Topics From Short Texts Such as Social Media Postings?

The evaluation results on tweets dataset, seen in Tables 10, and 14, make it clear that the Topic Distiller is not only able to identify topics in tweets, but that it exceeds the performance of state-of-the-art methods by a significant degree. Though the system was not evaluated on Facebook data, it is reasonable to assume that the Topic Distiller would be able to process Facebook status posts, along with other short social media postings, as well.

5.2.3. Is the Topic Distiller Able to Process Enough Documents Per Minute?

Figure 18 shows that all algorithms show significant increase in processing time when document length is increased and as the result, the number of documents processed per minute decreases. Conversely, the number of extracted keyphrases does not seem to affect the processing time immensly in any of the algorithms. Though not the swiftest algorithm available, Topic Distiller is able to process a reasonable amount of documents per minute.

5.2.4. Does Machine Translation Have an Effect on the Output of the Topic Distiller?

Tables 20, 21, and 22 show that using machine translation significantly decreases the performance of the Topic Distiller. This is evident from decrease of all performance metrics when machine translated documents are compared to the original ones. That being said, the Topic Distiller was still able to identify some of the original keyphrases using the translated documents making it at least a reasonable fall-back if no other methods are found for non-English texts.

5.3. Future Work

The current version of the Topic Distiller has three tunable parameters: *the number of hops* used to traverse the DBpedia graph (see Section: 3.3.2), and the coefficients α and β (see Section: 3.3.4), former of which is used to weight the importance of degree centrality in ranking and latter for the importance of topic overlap score. However, the effect of tuning these parameters was not evaluated. This calls for further analysis in later work.

The current version of the Topic Distiller is only able to process data in English language. The authors of DBpedia provide the ability to compile DBpedia with all languages available in Wikipedia. The performance of the Topic Extracor using different language datasets should be researched. The integration of ontologies apart from DBpedia, such as Wikidata and Yago, should also be investigated.

An interesting research problem would be to combine the power of neural networks with the vast knowledge of ontologies for AKE. The combination of neural networks to knowledge graphs have already been researched and systems such as Graves et al.'s *Differentiable Neural Computer* [101] and Pham et al.'s *Graph Memory Networks* [102] show very promising results. These neural networks are able to *reason* by learning how to traverse konwledge graphs. This reasoning ability would most likely be beneficial for AKE as well. Using deep learning to connect deep neural networks to knowledge bases should not be confused with the deep learning metrics discussed in Section 2.3.1, which focus only on extracting or generating keyphrases from text instead of using knowledge bases as basis for logical reasoning. Alas, the lack of large datasets also prohibit these methods to reach their potential. Deep learning has been very successful in the area of machine vision where huge datasets of labeled data, such as ImageNet[1], are freely available. Future work should be directed towards building such datasets for natural language processing tasks, such as AKE, as well.

[1] http://www.image-net.org/

6. CONCLUSION

The goal of this thesis was to build a system able to extract and identify semantic keyphrases, or topics, from textual documents. The developed system was named Topic Distiller, for its ability to reduce documents to their topics.

To develop such as system, research on modern AKE methods was conducted. It soon become evident, that unsupervised AKE methods are the best suited for the system developed in this thesis. This is due to the fact that there are no big datasets of documents with annotated keyphrases. The only freely available datasets are quite small in modern standards and they only contain articles of scientific literature. Training supervised models on scientific articles would bias the model towards the style of scientific writing, hence making the model less robust for other types of text.

Graph-based methods are the state-of-the-art in unsupervised AKE, but they lack the ability to identify topics not present in the documents fed to them. To tackle this problem, inspiration was drawn from the discipline of ATL. This resulted in employing external knowledge in the form of ontologies, such as DBpedia in this case, to give the system the ability to use information not present in the document being processed. Though far from perfect, this ability sets the Topic Distiller apart from conventional AKE methods. This enhancement was not without cost; the system implemented now has to rely on third party software.

Modern AKE methods also suffer from the issue of sometimes producing nonsensical keyphrases, since they use lexical methods to manipulate the words and sentences in the documents to generate the keyphrase candidates. For this reason we argue in this thesis that it is better to use EL to create the list of candidates. Employing EL to this task ensures that all keyphrase candidates represent real world phenomena, since all entities linked can be found from the knowledge base. Using lexical methods for keyphrase candidate generation has the advantage over EL of being able produce keyphrases that are novel or obscure. For some applications this could mean that using EL is insufficient and should be supplemented with lexical methods. However, for the purposes of this thesis using EL alone is satisfactory.

Special interest was given to the ability of the system to analyze social media postings as they are an overwhelmingly plentiful source of information. Social media posting, such as tweets, are infamous for being difficult to extract knowledge from and many of the current AKE methods fail to do so. This thesis shows that the Topic Distiller is not only able to discover topics discussed in tweets, but does so with unparalleled efficacy.

To implement the design (see Section 3), the Python programming language was adopted. This decision was easy since Python is free and open-source with a large community of scientific computing behind it. This community contributes good libraries providing functionality required in the making of the Topic Distiller. Using open-source software minimized the amount of code that needed to be developed for the Topic Distiller making future maintenance a lesser task.

To ensure that the Topic Distiller satisfies the requirements set for it in Section 3.1, an evaluation was conducted (see Section 4). The results of the evaluation (see Section 4.5) show that the system is able to perform comparably to state-of-the-art AKE methods when scientific text is in question and outperforms them when evaluated with news articles and social media postings. The processing time evaluation indicates

that the Topic Distiller, while not the fastest available method, is able to process data with reasonable velocity.

The current version of the Topic Distiller can only process text written in English, thus limiting its usage in other languages. An evaluation was carried out to investigate the effect of machine translation on the performance of the Topic Distiller (see Sections 4.4 and 4.5.5) revealing that machine translation negatively affects said performance. Luckily, most of the content residing in the world wide web is written in English making the Topic Distiller usable for a wide range of natural language applications.

The Topic Distiller provides the simple service of finding semantic topics from text documents. A HTTP API was developed to present this service to other systems. Use cases of the Topic Distiller include the analysis of trending topics in social media, summarization, document indexing, and semantic search.

7. REFERENCES

[1] Berners-Lee T., Hendler J. & Lassila O. (2001) The semantic web. Scientific american 284, pp. 34–43.

[2] Auer S., Bizer C., Kobilarov G., Lehmann J., Cyganiak R. & Ives Z. (2007) Dbpedia: A nucleus for a web of open data. In: The semantic web, Springer, pp. 722–735.

[3] Suchanek F.M., Kasneci G. & Weikum G. (2007) Yago: a core of semantic knowledge. In: Proceedings of the 16th international conference on World Wide Web, ACM, pp. 697–706.

[4] Nguyen C.Q. & Phan T.T. (2009) An ontology-based approach for key phrase extraction. In: Proceedings of the ACL-IJCNLP 2009 Conference Short Papers, Association for Computational Linguistics, pp. 181–184.

[5] Allahyari M Pouriyeh S K.K. & HR A. (2009), OntoLDA: An ontology-based topic model for automatic topic labeling.

[6] Hulpus I., Hayes C., Karnstedt M. & Greene D. (2013) Unsupervised graph-based topic labelling using DBpedia. In: Proceedings of the sixth ACM international conference on Web search and data mining, ACM, pp. 465–474.

[7] Shadbolt N., Berners-Lee T. & Hall W. (2006) The semantic web revisited. IEEE intelligent systems 21, pp. 96–101.

[8] Bollacker K., Evans C., Paritosh P., Sturge T. & Taylor J. (2008) Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, AcM, pp. 1247–1250.

[9] (2018), http://opencyc.org/. URL: `http://opencyc.org/`.

[10] Miller E. (1998) An introduction to the resource description framework. Bulletin of the American Society for Information Science and Technology 25, pp. 15–19.

[11] Pérez J., Arenas M. & Gutierrez C. (2006) Semantics and complexity of SPARQL. In: International semantic web conference, Springer, pp. 30–43.

[12] Feigenbaum L. (2009), SPARQL by example. URL: `https://www.w3.org/2009/Talks/0615-qbe/`.

[13] Bray T., Paoli J., Sperberg-McQueen C.M., Maler E. & Yergeau F. (2000), Extensible markup language (XML) 1.0.

[14] Shen W., Wang J. & Han J. (2015) Entity linking with a knowledge base: Issues, techniques, and solutions. IEEE Transactions on Knowledge and Data Engineering 27, pp. 443–460.

[15] Grishman R. & Sundheim B. (1996) Message understanding conference-6: A brief history. In: COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics, vol. 1, vol. 1.

[16] Ji H. & Grishman R. (2011) Knowledge base population: Successful approaches and challenges. In: Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1, Association for Computational Linguistics, pp. 1148–1158.

[17] Bunescu R. & Paşca M. (2006) Using encyclopedic knowledge for named entity disambiguation. In: 11th conference of the European Chapter of the Association for Computational Linguistics.

[18] Mendes P.N., Jakob M., García-Silva A. & Bizer C. (2011) DBpedia spotlight: shedding light on the web of documents. In: Proceedings of the 7th international conference on semantic systems, ACM, pp. 1–8.

[19] Daiber J., Jakob M., Hokamp C. & Mendes P.N. (2013) Improving efficiency and accuracy in multilingual entity extraction. In: Proceedings of the 9th International Conference on Semantic Systems, ACM, pp. 121–124.

[20] Turney P.D. (2000) Learning algorithms for keyphrase extraction. Information retrieval 2, pp. 303–336.

[21] Das D. & Martins A.F. (2007) A survey on automatic text summarization. Literature Survey for the Language and Statistics II course at CMU 4, pp. 192–195.

[22] Lam W., Ruiz M. & Srinivasan P. (1999) Automatic text categorization and its application to text retrieval. IEEE Transactions on Knowledge and Data engineering 11, pp. 865–879.

[23] Berend G. (2011) Opinion expression mining by exploiting keyphrase extraction .

[24] Gutwin C., Paynter G., Witten I., Nevill-Manning C. & Frank E. (1999) Improving browsing in digital libraries with keyphrase indexes. Decision Support Systems 27, pp. 81–104.

[25] Hasan K.S. & Ng V. (2014) Automatic keyphrase extraction: A survey of the state of the art. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), vol. 1, vol. 1, pp. 1262–1273.

[26] Liu F., Pennell D., Liu F. & Liu Y. (2009) Unsupervised approaches for automatic keyword extraction using meeting transcripts. In: Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics, Association for Computational Linguistics, pp. 620–628.

[27] Mihalcea R. & Tarau P. (2004) Textrank: Bringing order into text. In: Proceedings of the 2004 conference on empirical methods in natural language processing.

[28] Huang C., Tian Y., Zhou Z., Ling C.X. & Huang T. (2006) Keyphrase extraction using semantic networks structure analysis. In: Data Mining, 2006. ICDM'06. Sixth International Conference on, IEEE, pp. 275–284.

[29] Kumar N. & Srinathan K. (2008) Automatic keyphrase extraction from scientific documents using N-gram filtration technique. In: Proceedings of the eighth ACM symposium on Document engineering, ACM, pp. 199–208.

[30] El-Beltagy S.R. & Rafea A. (2009) KP-Miner: A keyphrase extraction system for english and arabic documents. Information Systems 34, pp. 132–144.

[31] Newman D., Koilada N., Lau J.H. & Baldwin T. (2012) Bayesian text segmentation for index term identification and keyphrase extraction. Proceedings of COLING 2012 , pp. 2077–2092.

[32] Zhao W.X., Jiang J., He J., Song Y., Achananuparp P., Lim E.P. & Li X. (2011) Topical keyphrase extraction from Twitter. In: Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1, Association for Computational Linguistics, pp. 379–388.

[33] Marujo L., Ling W., Trancoso I., Dyer C., Black A.W., Gershman A., de Matos D.M., Neto J. & Carbonell J. (2015) Automatic keyword extraction on Twitter. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), vol. 2, vol. 2, pp. 637–643.

[34] Frank E., Paynter G.W., Witten I.H., Gutwin C. & Nevill-Manning C.G. (1999) Domain-specific keyphrase extraction. In: 16th International Joint Conference on Artificial Intelligence (IJCAI 99), vol. 2, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, vol. 2, pp. 668–673.

[35] Witten I.H., Paynter G.W., Frank E., Gutwin C. & Nevill-Manning C.G. (2005) KEA: Practical automated keyphrase extraction. In: Design and Usability of Digital Libraries: Case Studies in the Asia Pacific, IGI Global, pp. 129–152.

[36] Turney P.D. (2002) Learning to extract keyphrases from text. arXiv preprint cs/0212013 .

[37] Jiang X., Hu Y. & Li H. (2009) A ranking approach to keyphrase extraction. In: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, ACM, pp. 756–757.

[38] Salton G. & Buckley C. (1988) Term-weighting approaches in automatic text retrieval. Information processing & management 24, pp. 513–523.

[39] Kim S.N. & Kan M.Y. (2009) Re-examining automatic keyphrase extraction approaches in scientific articles. In: Proceedings of the workshop on multiword expressions: Identification, interpretation, disambiguation and applications, Association for Computational Linguistics, pp. 9–16.

[40] Nguyen T.D. & Kan M.Y. (2007) Keyphrase extraction in scientific publications. In: International Conference on Asian Digital Libraries, Springer, pp. 317–326.

[41] Yih W.t., Goodman J. & Carvalho V.R. (2006) Finding advertising keywords on web pages. In: Proceedings of the 15th international conference on World Wide Web, ACM, pp. 213–222.

[42] Barker K. & Cornacchia N. (2000) Using noun phrase heads to extract document keyphrases. In: Conference of the Canadian Society for Computational Studies of Intelligence, Springer, pp. 40–52.

[43] Hulth A. (2003) Improved automatic keyword extraction given more linguistic knowledge. In: Proceedings of the 2003 conference on Empirical methods in natural language processing, Association for Computational Linguistics, pp. 216–223.

[44] Medelyan O., Frank E. & Witten I.H. (2009) Human-competitive tagging using automatic keyphrase extraction. In: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3, Association for Computational Linguistics, pp. 1318–1327.

[45] Turney P.D. (2003) Coherent keyphrase extraction via web mining. arXiv preprint cs/0308033 .

[46] Zhang Q., Wang Y., Gong Y. & Huang X. (2016) Keyphrase extraction using deep recurrent neural networks on Twitter. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp. 836–845.

[47] Lipton Z.C., Berkowitz J. & Elkan C. (2015) A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019 .

[48] Basaldella M., Antolli E., Serra G. & Tasso C. (2018) Bidirectional LSTM recurrent neural network for keyphrase extraction. In: Italian Research Conference on Digital Libraries, Springer, pp. 180–187.

[49] Bougouin A., Boudin F. & Daille B. (2013) Topicrank: Graph-based topic ranking for keyphrase extraction. In: International Joint Conference on Natural Language Processing (IJCNLP), pp. 543–551.

[50] Meng R., Zhao S., Han S., He D., Brusilovsky P. & Chi Y. (2017) Deep keyphrase generation. arXiv preprint arXiv:1704.06879 .

[51] Cho K., Van Merriënboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk H. & Bengio Y. (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 .

[52] Sutskever I., Vinyals O. & Le Q.V. (2014) Sequence to sequence learning with neural networks. In: Advances in neural information processing systems, pp. 3104–3112.

[53] Page L., Brin S., Motwani R. & Winograd T. (1999) The PageRank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab.

[54] Florescu C. & Caragea C. (2017) PositionRank: An unsupervised approach to keyphrase extraction from scholarly documents. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), vol. 1, vol. 1, pp. 1105–1115.

[55] Wan X. & Xiao J. (2008) Single document keyphrase extraction using neighborhood knowledge. In: AAAI, vol. 8, vol. 8, pp. 855–860.

[56] Boudin F. (2018) Unsupervised keyphrase extraction with multipartite graphs. arXiv preprint arXiv:1803.08721 .

[57] Liu Z., Huang W., Zheng Y. & Sun M. (2010) Automatic keyphrase extraction via topic decomposition. In: Proceedings of the 2010 conference on empirical methods in natural language processing, Association for Computational Linguistics, pp. 366–376.

[58] Liu Z., Li P., Zheng Y. & Sun M. (2009) Clustering to find exemplar terms for keyphrase extraction. In: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1, Association for Computational Linguistics, pp. 257–266.

[59] Blei D.M., Ng A.Y. & Jordan M.I. (2003) Latent dirichlet allocation. Journal of machine Learning research 3, pp. 993–1022.

[60] Minka T. (2000), Estimating a Dirichlet distribution.

[61] Sterckx L., Demeester T., Deleu J. & Develder C. (2015) Topical word importance for fast keyphrase extraction. In: Proceedings of the 24th International Conference on World Wide Web, ACM, pp. 121–122.

[62] Grineva M., Grinev M. & Lizorkin D. (2009) Extracting key terms from noisy and multitheme documents. In: Proceedings of the 18th international conference on World wide web, ACM, pp. 661–670.

[63] Mani I. (2001) Automatic summarization, vol. 3. John Benjamins Publishing.

[64] Zha H. (2002) Generic summarization and keyphrase extraction using mutual reinforcement principle and sentence clustering. In: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, ACM, pp. 113–120.

[65] Wan X., Yang J. & Xiao J. (2007) Towards an iterative reinforcement approach for simultaneous document summarization and keyword extraction. In: Proceedings of the 45th annual meeting of the association of computational linguistics, pp. 552–559.

[66] Tomokiyo T. & Hurst M. (2003) A language model approach to keyphrase extraction. In: Proceedings of the ACL 2003 workshop on Multiword expressions: analysis, acquisition and treatment-Volume 18, Association for Computational Linguistics, pp. 33–40.

[67] Mikolov T., Chen K., Corrado G. & Dean J. (2013) Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 .

[68] Paltoglou G. & Thelwall M. (2013) More than bag-of-words: Sentence-based document representation for sentiment analysis. In: Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013, pp. 546–552.

[69] Braud C. & Denis P. (2015) Comparing word representations for implicit discourse relation classification. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 2201–2211.

[70] Pennington J., Socher R. & Manning C. (2014) GloVe: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 1532–1543.

[71] Bojanowski P., Grave E., Joulin A. & Mikolov T. (2016) Enriching word vectors with subword information. arXiv preprint arXiv:1607.04606 .

[72] Le Q. & Mikolov T. (2014) Distributed representations of sentences and documents. In: International Conference on Machine Learning, pp. 1188–1196.

[73] Mahata D., Kuriakose J., Shah R.R. & Zimmermann R. (2018) Key2Vec: Automatic ranked keyphrase extraction from scientific articles using phrase embeddings. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), vol. 2, vol. 2, pp. 634–639.

[74] Bennani-Smires K., Musat C., Jaggi M., Hossmann A. & Baeriswyl M. (2018) EmbedRank: Unsupervised keyphrase extraction using sentence embeddings. arXiv preprint arXiv:1801.04470 .

[75] Matsuo Y. & Ishizuka M. (2004) Keyword extraction from a single document using word co-occurrence statistical information. International Journal on Artificial Intelligence Tools 13, pp. 157–169.

[76] Buckley C. & Voorhees E.M. (2004) Retrieval evaluation with incomplete information. In: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, ACM, pp. 25–32.

[77] Zesch T. & Gurevych I. (2009) Approximate matching for evaluating keyphrase extraction. In: Proceedings of the International Conference RANLP-2009, pp. 484–489.

[78] Li Z., Zhou D., Juan Y.F. & Han J. (2010) Keyword extraction for social snippets. In: Proceedings of the 19th international conference on World wide web, ACM, pp. 1143–1144.

[79] Haghighi A. & Vanderwende L. (2009) Exploring content models for multi-document summarization. In: Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, pp. 362–370.

[80] Titov I. & McDonald R. (2008) Modeling online reviews with multi-grain topic models. In: Proceedings of the 17th international conference on World Wide Web, ACM, pp. 111–120.

[81] Brody S. & Lapata M. (2009) Bayesian word sense induction. In: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, pp. 103–111.

[82] Dumais S.T., Furnas G.W., Landauer T.K., Deerwester S. & Harshman R. (1988) Using latent semantic analysis to improve access to textual information. In: Proceedings of the SIGCHI conference on Human factors in computing systems, Acm, pp. 281–285.

[83] Hofmann T. (1999) Probabilistic latent semantic analysis. In: Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence, Morgan Kaufmann Publishers Inc., pp. 289–296.

[84] Lau J.H., Grieser K., Newman D. & Baldwin T. (2011) Automatic labelling of topic models. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1, Association for Computational Linguistics, pp. 1536–1545.

[85] Mei Q., Shen X. & Zhai C. (2007) Automatic labeling of multinomial topic models. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp. 490–499.

[86] Grieser K., Baldwin T., Bohnert F. & Sonenberg L. (2011) Using ontological and document similarity to estimate museum exhibit relatedness. Journal on Computing and Cultural Heritage (JOCCH) 3, p. 10.

[87] Bhatia S., Lau J.H. & Baldwin T. (2016) Automatic labelling of topics with neural embeddings. arXiv preprint arXiv:1612.05340 .

[88] Kleinberg J.M. (1999) Hubs, authorities, and communities. ACM computing surveys (CSUR) 31, p. 5.

[89] Hulpus I., Hayes C. & Greene D. An eigenvalue-based measure for word-sense. computer 1, p. 1.

[90] Aletras N. & Stevenson M. (2014) Labelling topics using unsupervised graph-based methods. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), vol. 2, vol. 2, pp. 631–636.

[91] Knuth D.E. (1974) Structured programming with go to statements. ACM Computing Surveys (CSUR) 6.4 , pp. 261–301.

[92] Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P. & Berners-Lee T. (1999) Hypertext transfer protocol–HTTP/1.1. Tech. rep.

[93] Bray T. (2017) The JavaScript object notation (JSON) data interchange format. Tech. rep.

[94] Boudin F. (2013) A comparison of centrality measures for graph-based keyphrase extraction. In: International Joint Conference on Natural Language Processing (IJCNLP), pp. 834–838.

[95] (2018), spaCy · industrial-strength natural language processing in Python. URL: https://spacy.io/.

[96] (2018), aiosparql · PyPi. URL: https://pypi.org/project/aiosparql/.

[97] (2018), NetworkX — NetworkX. URL: https://networkx.github.io/.

[98] (2018), Welcome | Flask (A Python Microframework). URL: http://flask.pocoo.org/.

[99] Kim S.N., Medelyan O., Kan M.Y., Baldwin T. & Pingar L. SemEval-2010 Task 5: Automatic keyphrase extraction from scientific articles .

[100] Boudin F. (2016) pke: an open source python-based keyphrase extraction toolkit. In: Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations, pp. 69–73.

[101] Graves A., Wayne G., Reynolds M., Harley T., Danihelka I., Grabska-Barwińska A., Colmenarejo S.G., Grefenstette E., Ramalho T., Agapiou J. et al. (2016) Hybrid computing using a neural network with dynamic external memory. Nature 538, p. 471.

[102] Pham T., Tran T. & Venkatesh S. (2018) Graph memory networks for molecular activity prediction. arXiv preprint arXiv:1801.02622 .

8. APPENDIX

8.1. Composing Datasets

To compose the guardian and wikinews (see Section: 4.1.1) datasets their respective web sites[1][2] were crawled using a web crawling robot. Both sites have `robots.txt` files[3][4] for providing rules for web crawling robots. The rules are divided to different user agents. We used the rules for user agent when crawling the sites. Neither the Guardian or Wikinews require a minimum interval between downloads (Crawl-delay), but Wikinews had the following section in their `robots.txt`:

> Friendly, low-speed bots are welcome viewing article pages, but not dynamically-generated pages please.

We used the Crawl delay of 1000 milliseconds to ensure ethical use of these free resources.

Creating the tweets dataset (see Section: 4.1.1) did not require web scraping since the official Twitter API[5] was used.

The following sections contain more detailed descriptions of how the raw data for the datasets was gathered.

8.1.1. Guardian

The guardian dataset was built by scraping articles from URLs starting with `https://www.theguardian.com/world/2018/aug/`. These URLs contain world news published by the Guardian in August of 2018. From these articles 190 were selected randomly to be included in the dataset. None of the scraped URLs are included in the list of disallowed pages to scraper according to the `robots.txt` provided by the Guardian news web site.

8.1.2. Tweets

The tweets dataset was built using the *Filter realtime Tweets* (FRT) API[6] provided by Twitter. This API allows the streaming of real-time twitter statuses or tweets using keywords. The keywords are used to filter the incoming tweets so that only the tweets that include one ore more of those keywords are included in the stream. The FRT API also allows language to be used in the filtering of the tweet stream. To collect the tweets composing the tweets dataset we used the keywords *trump*, *brexit*, *business*, *ai*, *music*, and *emmys*, and filtered the tweets using the language English.

[1] https://www.theguardian.com/
[2] https://en.wikinews.org/
[3] https://www.theguardian.com/robots.txt
[4] https://en.wikinews.org/robots.txt
[5] https://developer.twitter.com/
[6] https://developer.twitter.com/en/docs/tweets/filter-realtime/api-reference/post-statuses-filter.html

8.1.3. Wikinews

To build the Wikinews dataset the articles in the Wikinews archive[7] were crawled and news containing versions both in English and Portuguese were used.

[7]https://en.wikinews.org/wiki/Wikinews:Archives/Date/2018